Unit-1: Concepts of Android and Setting up Android Environment

1.1 Introduction of Android

Android is a complete set of software for mobile devices such as tablet computers, notebooks, smartphones, electronic book readers, set-top boxes etc.

It contains a **linux-based Operating System**, **middleware** and **key mobile applications**.

It can be thought of as a **mobile operating system.**

But it is not limited to mobile only. It is currently used in various devices such as mobiles, tablets, televisions etc.

What is Android?

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

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

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

What is Open Handset Alliance (OHA)?

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

It was established on 5th November, 2007, led by Google.

It is committed to advance open standards, provide services and deploy handsets using the Android Plateform.

History and its Different Version List

The code names of android ranges from A to J currently, such as **Aestro**, **Blender**, **Cupcake**, **Donut**, **Eclair**, **Froyo**, **Gingerbread**, **Honeycomb**, **Ice Cream**

Sandwich, **Jelly Bean**, **KitKat** and **Lollipop**. Let's understand the android history in a sequence.

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

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

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

4) Originally intended for camera but shifted to smart phones later because of low market for camera only.

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

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

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

What is API level?

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

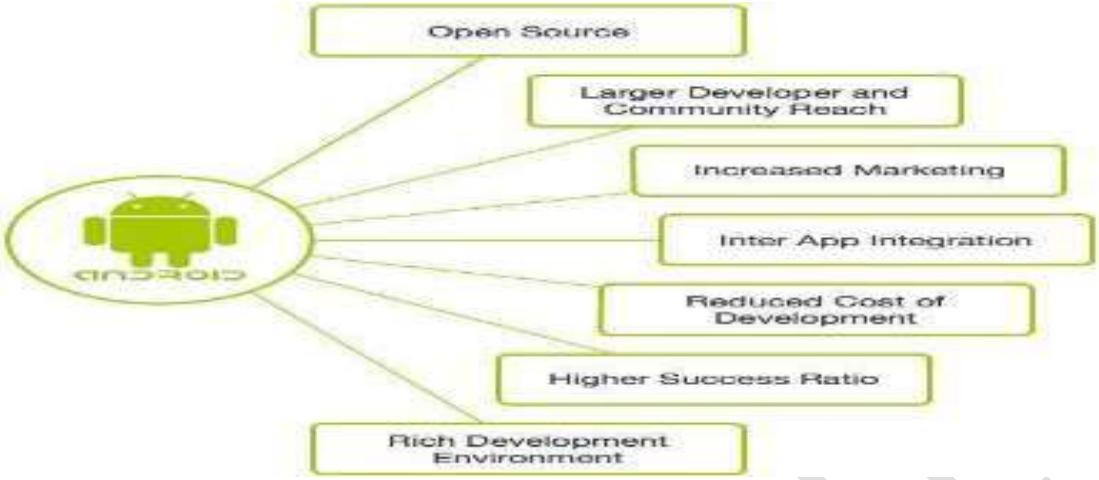
Version	Code name	API Level
1.5	Cupcake	3
1.6	Donut	4
2.1	Éclair	7

2.2	Froyo	8
2.3	Gingerbread	9 and 10
3.1 and 3.3	Honeycomb	12 and 13
4.0	Ice Cream Sandwich	15
4.1, 4.2 and 4.3	Jelly Bean	16, 17 and 18
4.4	KitKat	19
5.0	Lollipop	21
6.0	Marshmallow	23
7.0	Nougat	24-25
8.0	Oreo	26-27

Android Versions, Codename and API

Let's see the android versions, codenames and API Level provided by Google.

Why Android ?



Features of Android

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

Sr.No.	Feature & Description
1	**Beautiful UI** Android OS basic screen provides a beautiful and intuitive user interface.
2	**Connectivity** GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
3	**Storage** SQLite, a lightweight relational database, is used for data storage purposes.
4	**Media support** H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP.

5	**Messaging** SMS and MMS
6	**Web browser** Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.
7	**Multi-touch** Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.
8	**Multi-tasking** User can jump from one task to another and same time various application can run simultaneously.
9	**Resizable widgets** Widgets are resizable, so users can expand them to show more content or shrink them to save space.
10	**Multi-Language** Supports single direction and bi-directional text.
11	**GCM** Google Cloud Messaging (GCM) is a service that lets developers send short message data to their users on Android devices, without needing a proprietary sync solution.
12	**Wi-Fi Direct** A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection.

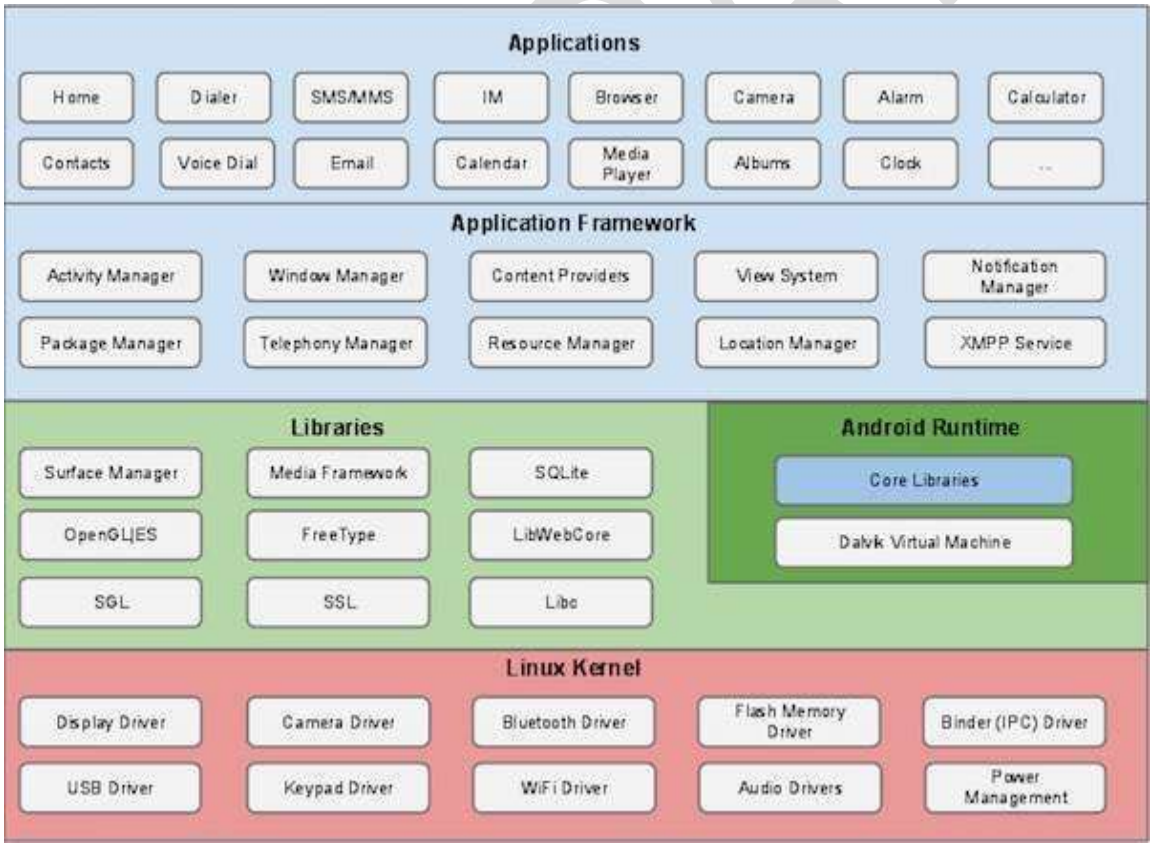
13	**Android Beam** A popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together.

1.1 Introduction of Android Architecture

OR

Software Stack

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.



Linux kernel

At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches.

This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc.

Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

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.

Android Libraries

This category encompasses those Java-based libraries that are specific to Android development.

Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access.

A summary of some key core Android libraries available to the Android developer is as follows −

- **android.app** − Provides access to the application model and is the cornerstone of all Android applications.
- **android.content** − Facilitates content access, publishing and messaging between applications and application components.
- **android.database** − Used to access data published by content providers and includes SQLite database management classes.
- **android.opengl** − A Java interface to the OpenGL ES 3D graphics rendering API.
- **android.os** − Provides applications with access to standard operating system services including messages, system services and inter-process communication.
- **android.text** − Used to render and manipulate text on a device display.
- **android.view** − The fundamental building blocks of application user interfaces.

- **android.widget** − A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
- **android.webkit** − A set of classes intended to allow web-browsing capabilities to be built into applications.

Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

Android Runtime

This is the third section of the architecture and available on the second layer from the bottom.

This section provides a key component called **Dalvik Virtual Machine** which is a kind of Java Virtual Machine specially designed and optimized for Android.

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.

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes.

Application developers are allowed to make use of these services in their applications.

The Android framework includes the following key services −

- **Activity Manager** − Controls all aspects of the application lifecycle and activity stack.
- **Content Providers** − Allows applications to publish and share data with other applications.

- **Resource Manager** − Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- **Notifications Manager** − Allows applications to display alerts and notifications to the user.
- **View System** − An extensible set of views used to create application user interfaces.

Applications

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

Dalvik Virtual Machine | DVM

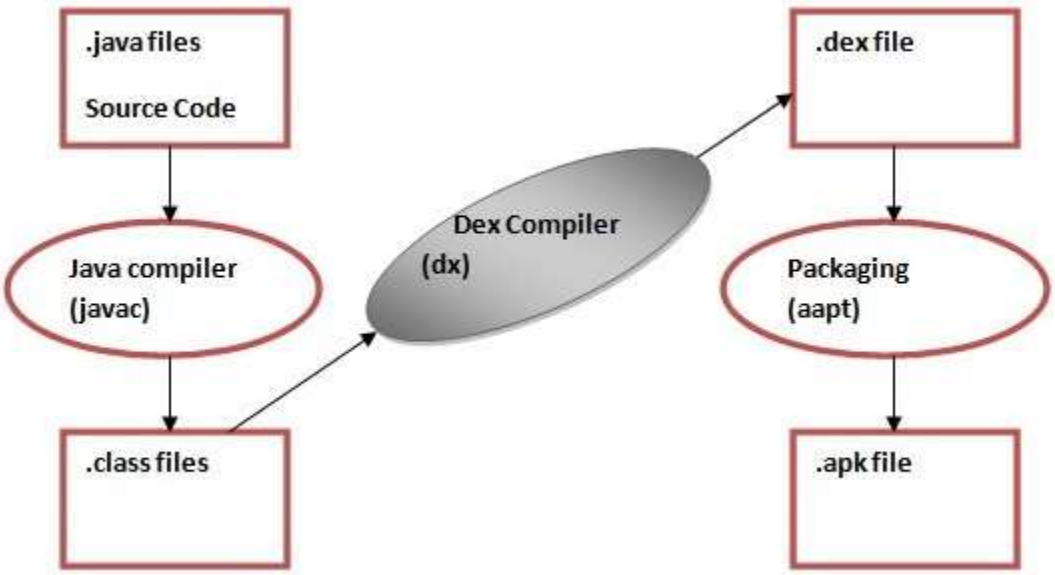
As we know the modern JVM is high performance and provides excellent memory management. But it needs to be optimized for low-powered handheld devices as well.

The **Dalvik Virtual Machine (DVM)** is an android virtual machine optimized for mobile devices. It optimizes the virtual machine for memory, battery life and performance.

Dalvik is a name of a town in Iceland. The Dalvik VM was written by Dan Bornstein.

The Dex compiler converts the class files into the .dex file that run on the Dalvik VM. Multiple class files are converted into one dex file.

Let's see the compiling and packaging process from the source file:



The **javac tool** compiles the java source file into the class file.

The **dx tool** takes all the class files of your application and generates a single .dex file. It is a platform-specific tool.

The **Android Assets Packaging Tool (aapt)** handles the packaging process.

1.4 Android Emulator

I .4. I Setting up JDK and Android Studio
I.4.2 Android SDK`manager

1. Setting up JDK (Java Development Kit)

Before installing Android Studio, you need to install the correct version of JDK on your system.

Steps:

1. **Download and Install JDK:**
    1. Go to the Oracle JDK download page or the OpenJDK website to get the JDK.
    2. Download the appropriate JDK version for your operating system.
    3. Install the JDK by following the installation prompts.
2. **Verify JDK Installation:**

    1. After installation, open your terminal or command prompt and type the following command to ensure the JDK is installed correctly:

    java -version

You should see the version of the JDK you installed displayed in the terminal.

2. Setting up Android Studio

Steps:

1.

Download Android Studio:

2.
1. Go to the official Android Studio website.
2. Download the appropriate version for your operating system.
3.

Install Android Studio:

4.

1. Run the installer and follow the prompts to install Android Studio on your machine.
2. During installation, Android Studio will ask if you want to install the Android SDK. Make sure to check that option.

3. Android SDK Manager (I.4.2)

The SDK Manager allows you to download and manage different versions of the Android SDK (Software Development Kit) and related tools. You can also use it to install system images for emulators, platform tools, and more.

Accessing the SDK Manager:

Open SDK Manager from Android Studio:

1. Launch Android Studio.
2. Go to the top menu and select **Tools** > **SDK Manager**.

Select SDK Packages:

1. In the SDK Manager window, you will see a list of SDK platforms and tools.
2. Check the boxes for the SDK versions and tools you want to install. Typically, you want to install the latest SDK version along with the **Android SDK Platform-Tools** and **Android SDK Build-Tools**.

I.5 Creating Android Virtual Device (AVD)

An **Android Virtual Device (AVD)** is an emulator configuration that allows you to simulate a variety of Android devices on your computer, making it easier to test and debug Android applications.

Here's a step-by-step guide on creating an Android Virtual Device (AVD) in Android Studio:

1. Open the AVD Manager

- Launch **Android Studio**.
- Go to the **top menu** and select **Tools** > **AVD Manager**.
    - Alternatively, you can find the **AVD Manager** icon on the toolbar. It looks like a small phone with the Android logo on it.

2. Create a New Virtual Device

- Once the **AVD Manager** window is open, click on the **Create Virtual Device...** button.

3. Select a Hardware Profile

You will be prompted to select a hardware profile. Android Studio provides a range of predefined devices such as:

- **Pixel devices** (e.g., Pixel 5, Pixel 6)
- **Nexus devices**
- Tablets, WearOS, or TV devices

Select the device type that closely matches the real hardware you want to emulate. You can also create a custom hardware profile if needed.

Click **Next**.

4. Choose a System Image

- The next step is to select the Android version (also called the **system image**) for the virtual device.
    - You can choose from different **API levels** (e.g., Android 13 (API level 33), Android 12 (API level 31)).
- The system image can be based on various architectures (x86, x86_64, ARM). If your computer supports it, choose an x86 image for better performance.
    - If the system image isn't downloaded, you'll see a **Download** link next to the image. Click it to download.
- Once you've selected the system image, click **Next**.

5. Configure the AVD Settings

- Now you can configure the AVD by adjusting various settings such as:
    - **AVD Name**: Provide a custom name for the virtual device.
    - **Startup Orientation**: Choose between portrait or landscape orientation.
    - **Graphics Settings**: Select hardware or software acceleration based on your system's capability.
    - **Memory and Storage**: Configure the RAM and internal storage if necessary.
- Leave the defaults as they are unless you have specific requirements.

6. Verify the AVD Configuration

- Review your virtual device settings. If everything looks good, click **Finish**.

7. Launch the Virtual Device

- After the AVD is created, it will appear in the list of devices in the **AVD Manager**.
- Click the **Play** button (green triangle) next to your virtual device to launch the emulator.
- The emulator will start, and the virtual device will boot up, showing you the Android home screen.

8. Testing Your App on the AVD

- Once the AVD is running, you can run your app directly on the virtual device.
- In **Android Studio**, simply click **Run** or **Debug** in the toolbar, and select your virtual device from the list of available devices.

Additional Options:

- **Modify AVD Settings**: You can edit an existing AVD by clicking the **Edit** button in the AVD Manager.
- **Wipe Data**: Reset the AVD to its default state by wiping data (e.g., if you want to start fresh with the device).
- **Snapshots**: You can take snapshots of the emulator's state and restore them later for faster booting and testing.

With these steps, you will have a functional Android Virtual Device (AVD) ready for app development and testing.

Install System Images for Emulators:

1. If you plan to use an Android emulator, scroll down to the **System Images** section.
2. Select the appropriate system image based on the API level and architecture (e.g., ARM, x86).

Apply Changes:

1. After selecting the components you want to install, click **Apply** or **OK**.
2. The SDK Manager will download and install the selected SDK packages.

With the JDK, Android Studio, and SDK set up, you'll be ready to develop Android apps or run Android emulators for testing purposes.

Unit-2: Creating basic App

2.1 Creating Basic App

2.1.1 Activity

2.1.2 Layout

2.2 Basic App using Android studio

2.2.1 Create new android project

2.2.2 Write message and run

2.2.3 Understanding different components.

2.3 Understanding AndroidManifest.xml, R.java

How to make Basic android apps?

There are 3 steps to create a basic android apps as following:

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

You need to follow the 3 steps mentioned above for creating the android application.

1) Create the New Android project

For creating the new android studio project:

1) Select *Start a new Android Studio project*

2) Provide the following information: Application name, Company domain, Project location and Package name of application and click next.

3) Select the API level of application and click next.

4) Select the Activity type (Empty Activity).

5) Provide the Activity Name and click finish.

After finishing the Activity configuration, Android Studio auto generates the activity class and other required configuration files.

Now an android project has been created. You can explore the android project and see the simple program, it looks like this:

2) Write the message

File: activity_main.xml

Android studio auto generates code for activity_main.xml file. You may edit this file according to your requirement.

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.constraint.ConstraintLayout xmlns:android="
   http://schemas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     xmlns:tools="http://schemas.android.com/tools"
5.     android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    tools:context="first.javatpoint.com.welcome.MainActivity">

    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello Android!"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

    </android.support.constraint.ConstraintLayout>
    }
```

File: MainActivity.java

```
package first.javatpoint.com.welcome;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
10.             setContentView(R.layout.activity_main);
11.         }
12.     }
```

3) Run the android application

To run the android application, click the run icon on the toolbar or simply press Shift + F10.

The android emulator might take 2 or 3 minutes to boot. So please have patience. After booting the emulator, the android studio installs the application and launches the activity.

2.2.3 Android Core Building Blocks [Android Application Components]

An android **component** is simply a piece of code that has a well defined life cycle e.g. Activity, Receiver, Service etc.

The **core building blocks** or **fundamental components** of android are activities, views, intents, services, content providers, fragments and AndroidManifest.xml.

Activity

An activity is a class that represents a single screen. It is like a Frame in AWT.

View

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

Intent

Intent is used to invoke components. It is mainly used to:

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

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

```
1. Intent intent=new Intent(Intent.ACTION_VIEW);
2. intent.setData(Uri.parse("http://www.javatpoint.com"));
3. startActivity(intent);
```

Service

Service is a background process that can run for a long time.

There are two types of services local and remote. Local service is accessed from within the application whereas remote service is accessed remotely from other applications running on the same device.

Content Provider

Content Providers are used to share data between the applications.

Fragment

Fragments are like parts of activity. An activity can display one or more fragments on the screen at the same time.

AndroidManifest.xml

It contains informations about activities, content providers, permissions etc. It is like the web.xml file in Java EE.

Android Virtual Device (AVD)

It is used to test the android application without the need for mobile or tablet etc. It can be created in different configurations to emulate different types of real devices.

2.3 Android Manifest File

Every project in Android includes a Manifest XML file, which is **AndroidManifest.xml**, located in the root directory of its project hierarchy.

The manifest file is an important part of our app because it defines the structure and metadata of our application, its components, and its requirements.

This file includes nodes for each of the **Activities**, **Services**, **Content Providers**, and **Broadcast Receivers** that make the application, and using **Intent Filters** and Permissions determines how they coordinate with each other and other applications.

The manifest file also specifies the application metadata, which includes its icon, version number, themes, etc., and additional top-level nodes can specify any required permissions, and unit tests, and define hardware, screen, or platform requirements.

The manifest comprises a root manifest tag with a package attribute set to the project's package. It should also include an xmls:android attribute that will supply several system attributes used within the file.

We use the **versionCode** attribute is used to define the current application version in the form of an integer that increments itself with the iteration of the version due to update.

Also, the **versionName** attribute is used to specify a public version that will be displayed to the users.

We can also specify whether our app should install on an SD card of the internal memory using the install Location attribute.

A typical manifest file looks as:

- **XML**

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

     <uses-sdk
       android:minSdkVersion="18"
       android:targetSdkVersion="27" />
```

```
<application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.MyApplication"
    tools:targetApi="31">
    <activity
      android:name=".MainActivity"
      android:exported="true">
 <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
   </activity>
  </application>
</manifest>
```

A manifest file includes the nodes that define the application components, security settings, test classes, and requirements that make up the application. Some of the manifest sub-node tags that are mainly used are:

1. manifest

The main component of the AndroidManifest.xml file is known as manifest. Additionally, the packaging field describes the activity class's package name. It must contain an <application> element with the xmlns:android and package attribute specified.

- XML

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

    </application>

</manifest>
```

2. uses-sdk

It is used to define a minimum and maximum SDK version by means of an API Level integer that must be available on a device so that our application functions properly, and the target SDK for which it has been designed using a combination of minSdkVersion, maxSdkVersion, and targetSdkVersion attributes, respectively. It is contained within the <manifest> element.

- XML

```
<uses-sdk
    android:minSdkVersion="18"
    android:targetSdkVersion="27" />
```

3. uses-permission

It outlines a system permission that must be granted by the user for the app to function properly and is contained within the <manifest> element. When an application is installed (on Android 5.1 and lower devices or Android 6.0 and higher), the user must grant the application permissions.

- XML

```
<uses-permission
    android:name="android.permission.CAMERA"
    android:maxSdkVersion="18" />
```

4. application

A manifest can contain only one application node. It uses attributes to specify the metadata for your application (including its title, icon, and theme). During development, we should include a debuggable attribute set to true to enable debugging, then be sure to disable it for your release builds. The application node also acts as a container for the Activity, Service, Content Provider, and Broadcast Receiver nodes that specify the application components. The name of our custom application class can be specified using the android:name attribute.

- XML

```
<application
```

```
    android:name=".GeeksForGeeks"
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@drawable/gfgIcon"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@android:style/Theme.Light"
    android:debuggable="true"
    tools:targetApi="31">

    <!-- application nodes -->

</application>
```

5. uses-library

It defines a shared library against which the application must be linked. This element instructs the system to add the library's code to the package's class loader. It is contained within the <application> element.

- XML

```
<uses-library
    android:name="android.test.runner"
```

```
    android:required="true" />
```

6. activity

The Activity sub-element of an application refers to an activity that needs to be specified in the AndroidManifest.xml file. It has various characteristics, like label, name, theme, launchMode, and others. In the manifest file, all elements must be represented by <activity>. Any activity that is not declared there won't run and won't be visible to the system. It is contained within the <application> element.

- XML

```
<activity
    android:name=".MainActivity"
    android:exported="true">
</activity>
```

7. intent-filter

It is the sub-element of activity that specifies the type of intent to which the activity, service, or broadcast receiver can send a response. It allows the component to receive intents of a certain type while filtering out those that are not useful for the component. The intent filter must contain at least one <action> element.

- XML

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

    <category android:name="android.intent.category.LAUNCHER" />
```

```
</intent-filter>
```

8. action

It adds an action for the intent-filter. It is contained within the <intent-filter> element.

- XML

```
<action android:name="android.intent.action.MAIN" />
```

9. category

It adds a category name to an intent-filter. It is contained within the <intent-filter> element.

- XML

```
<category android:name="android.intent.category.LAUNCHER" />
```

10. uses-configuration

The uses-configuration components are used to specify the combination of input mechanisms that are supported by our application. It is useful for games that require particular input controls.

- XML

```
<uses-configuration
  android:reqTouchScreen=”finger”
  android:reqNavigation=”trackball”
  android:reqHardKeyboard=”true”
  android:reqKeyboardType=”qwerty”/>
```

<**uses-configuration**

android:reqTouchScreen=”finger”

android:reqNavigation=”trackball”

android:reqHardKeyboard=”true”

android:reqKeyboardType=”twelvekey”/>

11. uses-features

It is used to specify which hardware features your application requires. This will prevent our application from being installed on a device that does not include a required piece of hardware such as NFC hardware, as follows:

- XML

<**uses-feature** android:name=”android.hardware.nfc” />

12. permission

It is used to create permissions to restrict access to shared application components. We can also use the existing platform permissions for this purpose or define your own permissions in the manifest.

- XML

<**permission**

android: name=”com.paad.DETONATE_DEVICE”

android:protectionLevel=“dangerous”

android:label=”Self Destruct”

```
    android:description="@string/detonate_description">
</permission>
```

2.3 Android R.java file

Android R.java is *an auto-generated file by aapt* (Android Asset Packaging Tool) that contains resource IDs for all the resources of res/ directory.

If you create any component in the activity_main.xml file, id for the corresponding component is automatically created in this file.

This id can be used in the activity source file to perform any action on the component.

Note: If you delete R.jar file, android creates it automatically.

Let's see the android R.java file. It includes a lot of static nested classes such as menu, id, layout, attr, drawable, string etc.

```
1. /* AUTO-GENERATED FILE.  DO NOT MODIFY.
2.  *
3.  * This class was automatically generated by the
4.  * aapt tool from the resource data it found.  It
5.  * should not be modified by hand.
6.  */
7.
8. package com.example.helloandroid;
9.
10.         public final class R {
```

```
11.         public static final class attr {
12.         }
13.         public static final class drawable {
14.             public static final int ic_launcher=0x7f020000;
15.         }
16.         public static final class id {
17.             public static final int menu_settings=0x7f070000;
18.         }
19.         public static final class layout {
20.             public static final int activity_main=0x7f030000;
21.         }
22.         public static final class menu {
23.             public static final int activity_main=0x7f060000;
24.         }
25.         public static final class string {
26.             public static final int app_name=0x7f040000;
27.             public static final int hello_world=0x7f040001;
28.             public static final int menu_settings=0x7f040002;
29.         }
30.         public static final class style {
31.             /**
32.             Base application theme, dependent on API level. This
    theme is replaced
```

```
33.             by AppBaseTheme from res/values-
    vXX/styles.xml on newer devices.
34.
35.
36.             Theme customizations available in newer API levels
    can go in
37.             res/values-
    vXX/styles.xml, while customizations related to
38.             backward-compatibility can go here.
39.
40.
41.         Base application theme for API 11+. This theme comp
    letely replaces
42.         AppBaseTheme from res/values/styles.xml on API 11+
     devices.
43.
44.     API 11 theme customizations can go here.
45.
46.         Base application theme for API 14+. This theme comp
    letely replaces
47.         AppBaseTheme from BOTH res/values/styles.xml and
48.         res/values-v11/styles.xml on API 14+ devices.
49.
50.     API 14 theme customizations can go here.
51.          */
```

```
52.             public static final int AppBaseTheme=0x7f050000;
53.             /**  Application theme.
54.        All customizations that are NOT specific to a particular AP
    I-level can go here.
55.              */
56.             public static final int AppTheme=0x7f050001;
57.          }
58.       }
```

Unit-3: XML (Extensible Markup Language)

3.1 Characteristic and Use of XML

What is xml?

- **Xml** (eXtensible Markup Language) is a mark up language.
- XML is designed to store and transport data.
- Xml was released in late 90's. it was created to provide an easy to use and store self describing data.
- XML became a W3C Recommendation on February 10, 1998.
- XML is not a replacement for HTML.
- XML is designed to be self-descriptive.
- XML is designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is platform independent and language independent.

Note: Self-describing data is the data that describes both its content and structure.

What is mark-up language?

A **mark up language** is a modern system for highlight or underline a document.

Students often underline or highlight a passage to revise easily, same in the sense of modern mark up language highlighting or underlining is replaced by tags.

Why xml?

Platform Independent and Language Independent:

The main benefit of xml is that you can use it to take data from a program like Microsoft SQL, convert it into XML then share that XML with other programs and platforms.

You can communicate between two platforms which are generally very difficult.

The main thing which makes XML truly powerful is its international acceptance.

Many corporation use XML interfaces for databases, programming, office application mobile phones and more.

It is due to its platform independent feature.

Features and Advantages of XML:

XML is widely used in the era of web development. It is also used to simplify data storage and data sharing.

The main features or advantages of XML are given below.

1) XML separates data from HTML

Unit-3: XML (Extensible Markup Language)

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.

With XML, data can be stored in separate XML files. This way you can focus on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML.

With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

2) XML simplifies data sharing

In the real world, computer systems and databases contain data in incompatible formats.

XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.

This makes it much easier to create data that can be shared by different applications.

3) XML simplifies data transport

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.

Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

4) XML simplifies Platform change

Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost.

XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

5) XML increases data availability

Different applications can access your data, not only in HTML pages, but also from XML data sources.

With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

6) XML can be used to create new internet languages

A lot of new Internet languages are created with XML.

Here are some examples:

- **XHTML**
- **WSDL** for describing available web services
- **WAP** and **WML** as markup languages for handheld devices
- **RSS** languages for news feeds
- **RDF** and **OWL** for describing resources and ontology
- **SMIL** for describing multimedia for the web

Unit-3: XML (Extensible Markup Language)

HTML vs XML

There are many differences between HTML (Hyper Text Markup Language) and XML (eXtensible Markup Language). The important differences are given below:

No.	HTML	XML
1)	HTML is used **to display data** and focuses on how data looks.	XML is a software and hardware independent tool used **to transport and store data**. It focuses on what data is.
2)	HTML is a **markup language** itself.	XML provides a **framework to define markup languages**.
3)	HTML is **not case sensitive**.	XML is **case sensitive**.
4)	HTML is a presentation language.	XML is neither a presentation language nor a programming language.
5)	HTML **has its own predefined tags**.	You **can define tags according to your need**.
6)	In HTML, it is **not necessary to use a closing tag**.	XML **makes it mandatory to use a closing tag**.
7)	HTML is **static** because it is used to display data.	XML is **dynamic** because it is used to transport data.
8)	HTML **does not preserve whitespaces**.	XML **preserve whitespaces**.

XML Attributes

XML elements can have attributes. By the use of attributes we can add the information about the element.

XML attributes enhance the properties of the elements.

Note: XML attributes must always be quoted. We can use single or double quote.

Let us take an example of a book publisher. Here, book is the element and publisher is the attribute.

1. `<book publisher="Tata McGraw Hill"></book>`
1. `<book publisher='Tata McGraw Hill'></book>`

Metadata should be stored as attribute and data should be stored as element.

1. `<book>`

2. <book category="computer">
3. <author> A & B </author>
4. </book>

Data can be stored in attributes or in child elements. But there are some limitations in using attributes, over child elements.

Why should we avoid XML attributes

- Attributes cannot contain multiple values but child elements can have multiple values.
- Attributes cannot contain tree structure but child element can.
- Attributes are not easily expandable. If you want to change in attribute's vales in future, it may be complicated.
- Attributes cannot describe structure but child elements can.
- Attributes are more difficult to be manipulated by program code.
- Attributes values are not easy to test against a DTD, which is used to define the legal elements of an XML document.

Difference between attribute and sub-element

In the context of documents, attributes are part of markup, while sub elements are part of the basic document contents.

In the context of data representation, the difference is unclear and may be confusing.

Same information can be represented in two ways:

1st way:

1. <book publisher="Tata McGraw Hill"> </book>

2nd way:

1. <book>
2. <publisher> Tata McGraw Hill </publisher>
3. </book>

In the first example publisher is used as an attribute and in the second example publisher is an element.

Both examples provide the same information but it is good practice to avoid attribute in XML and use elements instead of attributes.

XML Example

XML documents create a hierarchical structure looks like a tree so it is known as XML Tree that starts at "the root" and branches to "the leaves".

Example of XML Document

XML documents uses a self-describing and simple syntax:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set).

The next line describes the root element of the document (like saying: "this document is a note"):

```
<note>
```

The next 4 lines describe 4 child elements of the root (to, from, heading, and body).

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

And finally the last line defines the end of the root element.

```
</note>
```

XML documents must contain a **root element.** This element is "the parent" of all other elements.

Unit-3: XML (Extensible Markup Language)

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

All elements can have sub elements (child elements).

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).

All elements can have text content and attributes (just like in HTML).

Another Example of XML: Books

File: books.xml

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

Test it Now

The root element in the example is <bookstore>. All elements in the document are contained within <bookstore>.

The <book> element has 4 children: <title>,< author>, <year> and <price>.

Another Example of XML: Emails

File: emails.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<emails>
<email>
 <to>Vimal</to>
 <from>Sonoo</from>
 <heading>Hello</heading>
 <body>Hello brother, how are you!</body>
</email>
<email>
 <to>Peter</to>
 <from>Jack</from>
 <heading>Birth day wish</heading>
 <body>Happy birth day Tom!</body>
</email>
<email>
 <to>James</to>
 <from>Jaclin</from>
 <heading>Morning walk</heading>
 <body>Please start morning walk to stay fit!</body>
</email>
<email>
 <to>Kartik</to>
 <from>Kumar</from>
 <heading>Health Tips</heading>
 <body>Smoking is injurious to health!</body>
</email>
</emails>
```

XML Comments

XML comments are just like HTML comments. We know that the comments are used to make codes more understandable other developers.

Unit-3: XML (Extensible Markup Language)

XML Comments add notes or lines for understanding the purpose of an XML code. Although XML is known as self-describing data but sometimes XML comments are necessary.

Syntax

An XML comment should be written as:

```
<!-- Write your comment-->
```

You cannot nest one XML comment inside the another.

XML Comments Example

Let's take an example to show the use of comment in an XML example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--Students marks are uploaded by months-->
<students>
  <student>
    <name>Ratan</name>
    <marks>70</marks>
  </student>
  <student>
    <name>Aryan</name>
    <marks>60</marks>
  </student>
</students>
```

Rules for adding XML comments

- Don't use a comment before an XML declaration.
- You can use a comment anywhere in XML document except within attribute value.
- Don't nest a comment inside the other comment.

3.4 XML document:

3.4.1 Document Prolog Section

3.4.2 Document element section

XML – Documents

Unit-3: XML (Extensible Markup Language)

XML Document forms the full struture of xml formatted data composed with prolog and root element nested with other elements.

There would be only one root element, where root element encloses many other elements inside the final inner element holds the data.

XML Document can be divided as three components. They are:

1. **Prolog**
2. **Elements (Root or Other)**
3. **Data**

Below is the example of Customers XML document with the root element “customer_list”

```
<?xml version="1.0" encoding="UTF-8"?>
<customer_list>
        <customer>
                <name> Sanjay</name>
                <location> Mumbai</location>
        </customer>
        <customer>
                <name> Micheal</name>
                <location> Washington</location>
        </customer>
</customer_list>
```

In the above example,

Document Prolog — <?xml version=”1.0” encoding=”UTF-8”?>
Root Element — <customer_list>
Other Elements — <customer> , <name> , <location>
Data — Sanjay, Mumbai, Washington, Micheal

4. 1 Hiding Title bar

Android Hide Title Bar and Full Screen Example

The **requestWindowFeature(Window.FEATURE_NO_TITLE)** method of Activity must be called to hide the title. But, it must be coded before the setContentView method.

Code that hides title bar of activity

The **getSupportActionBar()** method is used to retrieve the instance of ActionBar class. Calling the hide() method of ActionBar class hides the title bar.

```
requestWindowFeature(Window.FEATURE_NO_TITLE);//will hide the title
getSupportActionBar().hide(); //hide the title bar
```

Code that enables full screen mode of activity

The **setFlags()** method of Window class is used to display content in full screen mode. You need to pass the **WindowManager.LayoutParams.FLAG_FULLSCREEN** constant in the setFlags method.

```
this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN); //show the activity in full screen
```

Android Hide Title Bar and Full Screen Example

activity_main.xml

File: activity_main.xml

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

    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello World!"
```

```
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    </android.support.constraint.ConstraintLayout>
```

Activity class

File: MainActivity.java

```
package first.javatpoint.com.hidetitlebar;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Window;
import android.view.WindowManager;

public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE); //will hide the title
    getSupportActionBar().hide(); // hide the title bar
    this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN); //enable full screen
    setContentView(R.layout.activity_main);


  }
}
```

Output:

4.2 screen Orintation (Portrait, Landscape)

Android Screen Orientation Example

The **screenOrientation** is the attribute of activity element. The orientation of android activity can be portrait, landscape, sensor, unspecified etc. You need to define it in the AndroidManifest.xml file.

Syntax:

```
<activity android:name="package_name.Your_ActivityName"
    android:screenOrientation="orirntation_type">
</activity>
```

Example:

```
<activity android:name=" example.javatpoint.com.screenorientation.MainActivity"
    android:screenOrientation="portrait">
</activity>

<activity android:name=".SecondActivity"
    android:screenOrientation="landscape">
</activity>
```

The common values for screenOrientation attribute are as follows:

Value	Description
unspecified	It is the default value. In such case, system chooses the orientation.
portrait	taller not wider
landscape	wider not taller
sensor	orientation is determined by the device orientation sensor.

Android Portrait and Landscape mode screen orientation example

In this example, we will create two activities of different screen orientation. The first activity (MainActivity) will be as "portrait" orientation and second activity (SecondActivity) as "landscape" orientation type.

activity_main.xml

File: activity_main.xml

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

```
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.screenorientation.MainActivity">


    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginTop="112dp"
        android:onClick="onClick"
        android:text="Launch next activity"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.612"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText1"
        app:layout_constraintVertical_bias="0.613" />

    <TextView
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="124dp"
        android:ems="10"
        android:textSize="22dp"
        android:text="This activity is portrait orientation"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.502"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

Activity class

File: MainActivity.java

```
package example.javatpoint.com.screenorientation;

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

public class MainActivity extends AppCompatActivity {

  Button button1;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    button1=(Button)findViewById(R.id.button1);
  }
  public void onClick(View v) {
    Intent intent = new Intent(MainActivity.this,SecondActivity.class);
    startActivity(intent);
  }
}
```

activity_second.xml

File: activity_second.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/a
ndroid"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="example.javatpoint.com.screenorientation.SecondActivity">

  <TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="180dp"
    android:text="this is landscape orientation"
    android:textSize="22dp"
```

```
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.502"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    </android.support.constraint.ConstraintLayout>
```

SecondActivity class

File: SecondActivity.java

```
package example.javatpoint.com.screenorientation;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class SecondActivity extends AppCompatActivity {

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

  }
}
```

AndroidManifest.xml

File: AndroidManifest.xml
In AndroidManifest.xml file add the screenOrientation attribute in activity and provides its orientation. In this example, we provide "portrait" orientation for MainActivity and "landscape" for SecondActivity.

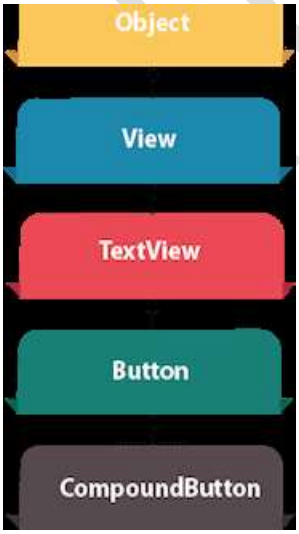
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="example.javatpoint.com.screenorientation">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity
      android:name="example.javatpoint.com.screenorientation.MainActivity"
      android:screenOrientation="portrait">
```

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

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

    </manifest>
```

Output:

4.3 .3 Form Widget Palette

4.3. I placing text fields and Button
4.3.2 Button onclick event



Android Button Example

Android Button represents a push-button. The android.widget.Button is subclass of TextView class and CompoundButton is the subclass of Button class.
There are different types of buttons in android such as RadioButton, ToggleButton, CompoundButton etc.

Android Button Example with Listener

Here, we are going to create two textfields and one button for sum of two numbers. If user clicks button, sum of two input values is displayed on the Toast.

We can perform action on button using different types such as calling listener on button or adding onClick property of button in activity's xml file.

```
1. button.setOnClickListener(new View.OnClickListener() {
2. @Override
3. public void onClick(View view) {
4. //code
5. }
6. });
```

```
1. <Button
2. android:onClick="methodName"
3. />
```

Drag the component or write the code for UI in activity_main.xml

First of all, drag 2 textfields from the Text Fields palette and one button from the Form Widgets palette as shown in the following figure.
The generated code for the ui components will be like this:

File: activity_main.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3. xmlns:app="http://schemas.android.com/apk/res-auto"
4. xmlns:tools="http://schemas.android.com/tools"

5. android:layout_width="match_parent"
```

```
6. android:layout_height="match_parent"
7. tools:context="example..com.sumoftwonumber.MainActivity">
8.
9. <EditText
10. android:id="@+id/editText1"
11. android:layout_width="wrap_content"
12. android:layout_height="wrap_content"
13. android:layout_alignParentTop="true"
14. android:layout_centerHorizontal="true"
15. android:layout_marginTop="61dp"
16. android:ems="10"
17. android:inputType="number"
18. tools:layout_editor_absoluteX="84dp"
19. tools:layout_editor_absoluteY="53dp" />
20.
21. <EditText
22. android:id="@+id/editText2"
23. android:layout_width="wrap_content"
24. android:layout_height="wrap_content"
25. android:layout_below="@+id/editText1"
26. android:layout_centerHorizontal="true"
27. android:layout_marginTop="32dp"
28. android:ems="10"
29. android:inputType="number"
30. tools:layout_editor_absoluteX="84dp"
31. tools:layout_editor_absoluteY="127dp" />
32.
33. <Button
34. android:id="@+id/button"
35. android:layout_width="wrap_content"
36. android:layout_height="wrap_content"
37. android:layout_below="@+id/editText2"
```

```
38. android:layout_centerHorizontal="true"
39. android:layout_marginTop="109dp"
40. android:text="ADD"
41. tools:layout_editor_absoluteX="148dp"
42. tools:layout_editor_absoluteY="266dp" />
43. </RelativeLayout>
```

Activity class

Now write the code to display the sum of two numbers.

File: MainActivity.java

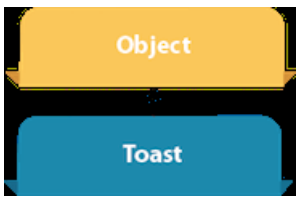
```
1. package example..com.sumoftwonumber;
2.
3. import android.support.v7.app.AppCompatActivity;
4. import android.os.Bundle;
5. import android.view.View;
6. import android.widget.Button;
7. import android.widget.EditText;
8. import android.widget.Toast;
9.
10. public class MainActivity extends AppCompatActivity {
11. private EditText edittext1, edittext2;
12. private Button buttonSum;
13.
14. @Override
15. protected void onCreate(Bundle savedInstanceState) {
16. super.onCreate(savedInstanceState);
17. setContentView(R.layout.activity_main);
18.
19. addListenerOnButton();
20. }
21.
22. public void addListenerOnButton() {

23. edittext1 = (EditText) findViewById(R.id.editText1);
```

```
24. edittext2 = (EditText) findViewById(R.id.editText2);
25. buttonSum = (Button) findViewById(R.id.button);
26.
27. buttonSum.setOnClickListener(new View.OnClickListener() {
28. @Override
29. public void onClick(View view) {
30. String value1=edittext1.getText().toString();
31. String value2=edittext2.getText().toString();
32. int a=Integer.parseInt(value1);
33. int b=Integer.parseInt(value2);
34. int sum=a+b;
35. Toast.makeText(getApplicationContext(),String.valueOf(sum),
Toast.LENGTH_LONG).show();
36. }
37. });
38. }
39. }
```

5.4 Displaying Notification:

5.4.1 Toast Class

5.4.2 Displaying message on Toast



Android Toast Example

Andorid Toast can be used to display information for the short period of time. A toast contains message to be displayed quickly and disappears after sometime.

The android.widget.Toast class is the subclass of java.lang.Object class.
You can also create custom toast as well for example toast displaying image. You can visit next page to see the code for custom toast.

Toast class

Toast class is used to show notification for a particular interval of time. After sometime it disappears. It doesn't block the user interaction.

Constants of Toast class

There are only 2 constants of Toast class which are given below. **Constant**	**Description**
public static final int LENGTH_LONG	displays view for the long duration of time.
public static final int LENGTH_SHORT	displays view for the short duration of time.

Methods of Toast class

The widely used methods of Toast class are given below.

Method	Description
public static Toast makeText(Context context, CharSequence text, int duration)	makes the toast containing text and duration.
public void show()	displays toast.
public void setMargin (float horizontalMargin, float verticalMargin)	changes the horizontal and vertical margin difference.

Android Toast Example

```
Toast.makeText(getApplicationContext(),"Hello Javatpoint",Toast.LENGTH_SHORT).show();
```

Another code:

```
Toast toast=Toast.makeText(getApplicationContext(),"Hello Javatpoint",Toast.LENGTH_SHORT);
toast.setMargin(50,50);
```

```
        toast.show();
```

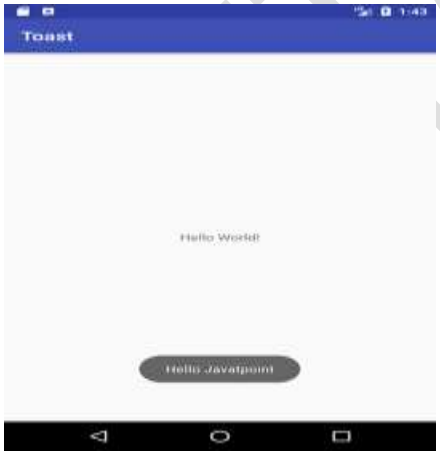
Note: Here, getApplicationContext() method returns the instance of Context.

Full code of activity class displaying Toast

File: MainActivity.java

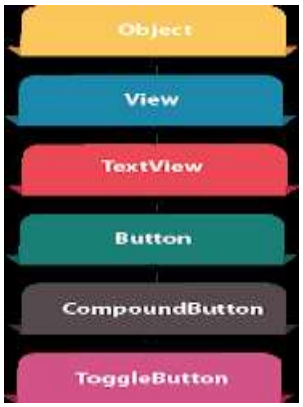
```
package example.javatpoint.com.toast;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

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

    //Displaying Toast with Hello Javatpoint message
    Toast.makeText(getApplicationContext(),"Hello Javatpoint",Toast.LENGTH_SHORT).show();
  }
}
```

Output:



4.5 TogglcButton:

4.5.1 ToggleButton Attributes:(textoff, texton)
4.5.2 Event methods : getTextoff(). getTexton(), setchecked()



Android ToggleButton Example

Android Toggle Button can be used to display checked/unchecked (On/Off) state on the button.

It is beneficial if user have to change the setting between two states. It can be used to On/Off Sound, Wifi, Bluetooth etc.

Since Android 4.0, there is another type of toggle button called *switch* that provides slider control.

Android ToggleButton and Switch both are the subclasses of CompoundButton class.

Android ToggleButton class

ToggleButton class provides the facility of creating the toggle button.

XML Attributes of ToggleButton class

There are 3 XML attributes of ToggleButton class.

XML Attribute	Description
android:disabledAlpha	The alpha to apply to the indicator when disabled.
android:textOff	The text for the button when it is not checked.
android:textOn	The text for the button when it is checked.

Methods of ToggleButton class

The widely used methods of ToggleButton class are given below.

Method	Description

CharSequence getTextOff()	Returns the text when button is not in the checked state.
CharSequence getTextOn()	Returns the text for when button is in the checked state.
void setChecked(boolean checked)	Changes the checked state of this button.

Android ToggleButton Example

activity_main.xml

Drag two toggle button and one button for the layout. Now the activity_main.xml file will look like this:

File: activity_main.xml

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

  <ToggleButton
    android:id="@+id/toggleButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="80dp"
    android:text="ToggleButton"
    android:textOff="Off"
    android:textOn="On"
    app:layout_constraintEnd_toStartOf="@+id/toggleButton2"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

  <ToggleButton
    android:id="@+id/toggleButton2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginRight="60dp"
    android:layout_marginTop="80dp"
    android:text="ToggleButton"
    android:textOff="Off"
    android:textOn="On"
```

```
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

  <Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="144dp"
    android:layout_marginLeft="148dp"
    android:text="Submit"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent" />
</android.support.constraint.ConstraintLayout>
```

Activity class

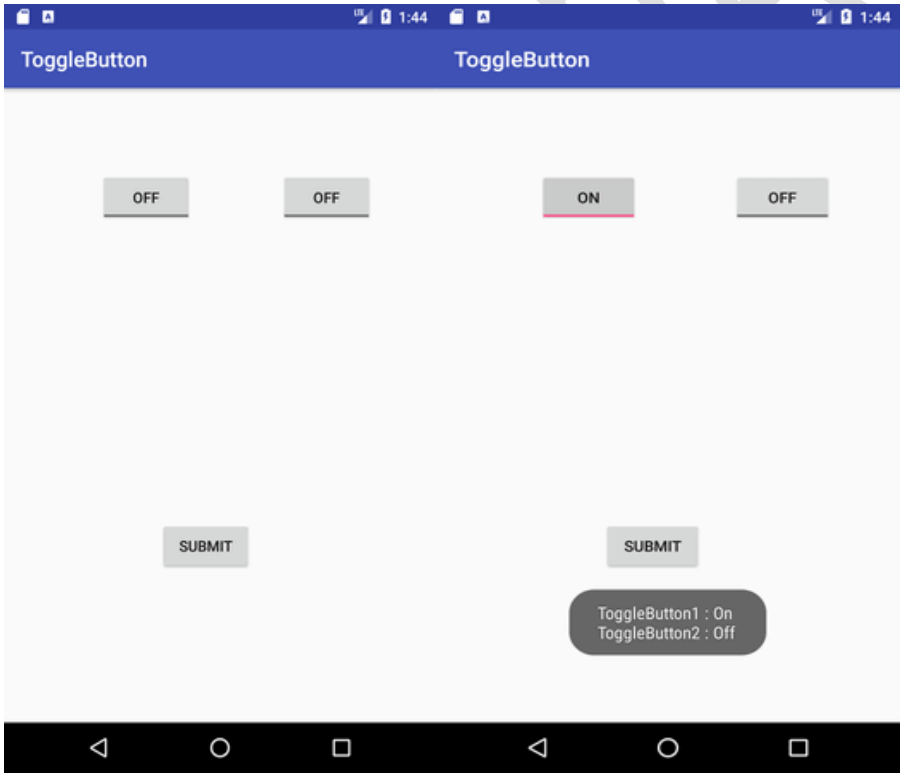
Let's write the code to check which toggle button is ON/OFF.

File: MainActivity.java

```
package example.javatpoint.com.togglebutton;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import android.widget.ToggleButton;

public class MainActivity extends AppCompatActivity {
  private ToggleButton toggleButton1, toggleButton2;
  private Button buttonSubmit;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    addListenerOnButtonClick();
  }

  public void addListenerOnButtonClick(){
    //Getting the ToggleButton and Button instance from the layout xml file
    toggleButton1=(ToggleButton)findViewById(R.id.toggleButton);
    toggleButton2=(ToggleButton)findViewById(R.id.toggleButton2);
    buttonSubmit=(Button)findViewById(R.id.button);
```

```
        //Performing action on button click
        buttonSubmit.setOnClickListener(new View.OnClickListener(){

            @Override
            public void onClick(View view) {
                StringBuilder result = new StringBuilder();
                result.append("ToggleButton1 : ").append(toggleButton1.getText());
                result.append("\nToggleButton2 : ").append(toggleButton2.getText());
                //Displaying the message in toast
                Toast.makeText(getApplicationContext(), result.toString(),Toast.LENGTH_LONG).show();
            }

        });

    }
}
```

Output:

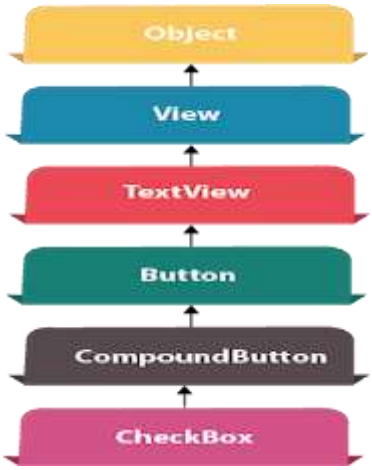


Unit-5: Other Android Widgets(UI)

5.1 CheckBox:

5. 1.1 Event methods: ischecked(), setchecked()

5. 1.2 Default and Custom Checkbox



Android CheckBox

Android provides facility to customize the UI of view elements rather than default.

You are able to create custom CheckBox in android. So, you can add some different images of checkbox on the layout.

Example of CheckBox

In this example, we create both default as well as custom checkbox. Add the following code in activity_main.xml file.

activity_main.xml

File: activity_main.xml

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

```

```
9.
10.   <TextView
11.      android:id="@+id/textView1"
12.     android:layout_width="fill_parent"
13.      android:layout_height="wrap_content"
14.     android:gravity="center_horizontal"
15.      android:textSize="25dp"
16.     android:text="Default Check Box"
17.      android:layout_alignParentTop="true"
18.     android:layout_alignParentLeft="true"
19.      android:layout_alignParentStart="true" />
20.
21.   <CheckBox
22.      android:layout_width="wrap_content"
23.      android:layout_height="wrap_content"
24.     android:text="New CheckBox"
25.      android:id="@+id/checkBox"
26.     android:layout_below="@+id/textView1"
27.      android:layout_centerHorizontal="true"
28.     android:layout_marginTop="46dp" />
29.
30.   <CheckBox
31.      android:layout_width="wrap_content"
32.     android:layout_height="wrap_content"
33.      android:text="New CheckBox"
34.     android:id="@+id/checkBox2"
35.      android:layout_below="@+id/checkBox"
36.     android:layout_alignLeft="@+id/checkBox"
37.      android:layout_alignStart="@+id/checkBox" />
38.
39.   <View
40.      android:layout_width="fill_parent"
41.      android:layout_height="1dp"
42.     android:layout_marginTop="200dp"
43.      android:background="#B8B894"
44.     android:id="@+id/viewStub" />
45.
46.   <CheckBox
47.      android:layout_width="wrap_content"
48.     android:layout_height="wrap_content"
49.      android:text="CheckBox 1"
50.     android:id="@+id/checkBox3"
51.      android:button="@drawable/customcheckbox"
52.     android:layout_below="@+id/viewStub"
53.      android:layout_centerHorizontal="true"
54.     android:layout_marginTop="58dp" />
55.
56.   <CheckBox
57.      android:layout_width="wrap_content"
```

```
58.       android:layout_height="wrap_content"
59.       android:text="CheckBox 2"
60.       android:id="@+id/checkBox4"
61.       android:button="@drawable/customcheckbox"
62.       android:layout_below="@+id/checkBox3"
63.       android:layout_alignLeft="@+id/checkBox3"
64.       android:layout_alignStart="@+id/checkBox3" />
65.
66.    <TextView
67.       android:layout_width="wrap_content"
68.       android:layout_height="wrap_content"
69.       android:textAppearance="?android:attr/textAppearanceSmall"
70.       android:textSize="25dp"
71.       android:text="Custom Check Box"
72.       android:id="@+id/textView"
73.       android:layout_alignTop="@+id/viewStub"
74.       android:layout_centerHorizontal="true" />
75.
76.    <Button
77.       android:layout_width="wrap_content"
78.       android:layout_height="wrap_content"
79.       android:text="Show Checked"
80.       android:id="@+id/button"
81.       android:layout_alignParentBottom="true"
82.       android:layout_centerHorizontal="true" />
83.
84. </RelativeLayout>
```

Now implement a selector in another file (checkbox.xml) under drawable folder which customizes the checkbox.

checkbox.xml

File: checkbox.xml

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <selector xmlns:android="http://schemas.android.com/apk/res/android">
3.     <item android:state_checked="true" android:drawable="@drawable/checked" />
4.     <item android:state_checked="false" android:drawable="@drawable/unchecked"/>
5.  </selector>
```

Activity class

File: MainActivity.java

```
1.  package example.javatpoint.com.customcheckbox;
2.
3.  import android.support.v7.app.AppCompatActivity;
4.  import android.os.Bundle;
5.  import android.view.View;
```

```
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    CheckBox cb1,cb2;
    Button button;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        cb1=(CheckBox)findViewById(R.id.checkBox3);
        cb2=(CheckBox)findViewById(R.id.checkBox4);
        button=(Button)findViewById(R.id.button);

        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                StringBuilder sb=new StringBuilder("");

                if(cb1.isChecked()){
                    String s1=cb1.getText().toString();
                    sb.append(s1);
                }

                if(cb2.isChecked()){
                    String s2=cb2.getText().toString();
                    sb.append("\n"+s2);

                }
                if(sb!=null && !sb.toString().equals("")){
                    Toast.makeText(getApplicationContext(), sb, Toast.LENGTH_LONG).show(
);

                }
                else{
                    Toast.makeText(getApplicationContext(),"Nothing Selected", Toast.LENGT
H_LONG).show();
                }

            }

        });
    }
}
```

Android CheckBox is a type of two state button either checked or unchecked.

There can be a lot of usage of checkboxes. For example, it can be used to know the hobby of the user, activate/deactivate the specific action etc.

Android CheckBox class is the subclass of CompoundButton class.

Android CheckBox class

The android.widget.CheckBox class provides the facility of creating the CheckBoxes.

Methods of CheckBox class

Method	Description
public boolean isChecked()	Returns true if it is checked otherwise false.
public void setChecked(boolean status)	Changes the state of the CheckBox.

There are many inherited methods of View, TextView, and Button classes in the CheckBox class. Some of them are as follows:

Android CheckBox Example

activity_main.xml

Drag the three checkboxes and one button for the layout. Now the activity_main.xml file will look like this:

File: activity_main.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     xmlns:tools="http://schemas.android.com/tools"
```

```
5.    android:layout_width="match_parent"
6.    android:layout_height="match_parent"
7.    tools:context="example.javatpoint.com.checkbox.MainActivity">
8.
9.
10.   <CheckBox
11.      android:id="@+id/checkBox"
12.      android:layout_width="wrap_content"
13.      android:layout_height="wrap_content"
14.      android:layout_marginLeft="144dp"
15.      android:layout_marginTop="68dp"
16.      android:text="Pizza"
17.      app:layout_constraintStart_toStartOf="parent"
18.      app:layout_constraintTop_toTopOf="parent" />
19.
20.   <CheckBox
21.      android:id="@+id/checkBox2"
22.      android:layout_width="wrap_content"
23.      android:layout_height="wrap_content"
24.      android:layout_marginLeft="144dp"
25.      android:layout_marginTop="28dp"
26.      android:text="Coffee"
27.      app:layout_constraintStart_toStartOf="parent"
28.      app:layout_constraintTop_toBottomOf="@+id/checkBox" />
29.
30.   <CheckBox
```

```
31.      android:id="@+id/checkBox3"
32.      android:layout_width="wrap_content"
33.      android:layout_height="wrap_content"
34.      android:layout_marginLeft="144dp"
35.      android:layout_marginTop="28dp"
36.      android:text="Burger"
37.      app:layout_constraintStart_toStartOf="parent"
38.      app:layout_constraintTop_toBottomOf="@+id/checkBox2" />
39.
40.   <Button
41.      android:id="@+id/button"
42.      android:layout_width="wrap_content"
43.      android:layout_height="wrap_content"
44.      android:layout_marginLeft="144dp"
45.      android:layout_marginTop="184dp"
46.      android:text="Order"
47.      app:layout_constraintStart_toStartOf="parent"
48.      app:layout_constraintTop_toBottomOf="@+id/checkBox3" />
49.
50.</android.support.constraint.ConstraintLayout>
```

Activity class

Let's write the code to check which toggle button is ON/OFF.

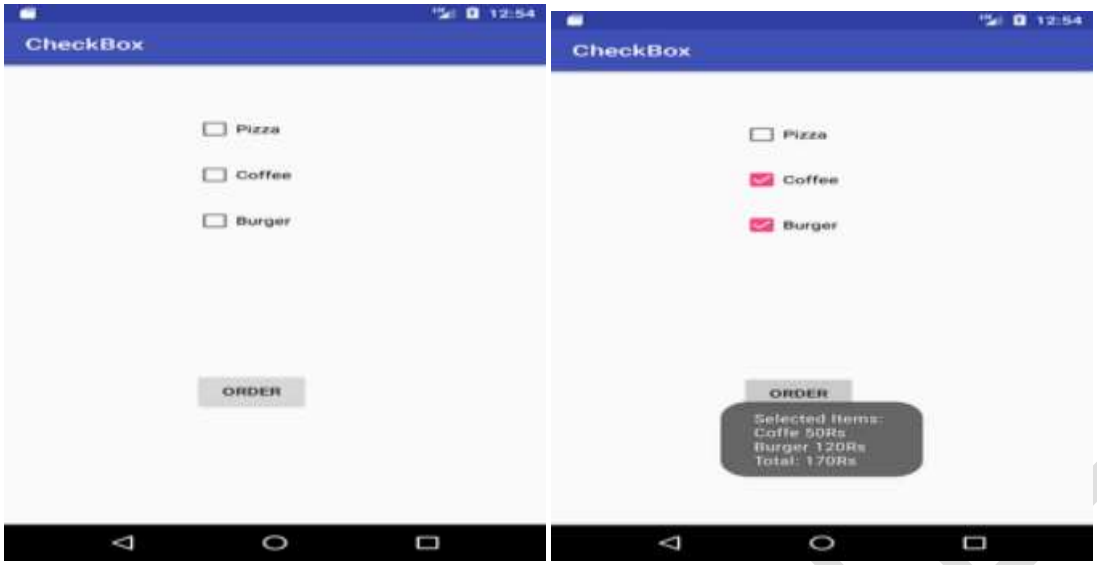
File: MainActivity.java

```
1. package example.javatpoint.com.checkbox;
2.
```

```
3. import android.support.v7.app.AppCompatActivity;
4. import android.os.Bundle;
5. import android.view.View;
6. import android.widget.Button;
7. import android.widget.CheckBox;
8. import android.widget.Toast;
9.
10.public class MainActivity extends AppCompatActivity {
11.    CheckBox pizza,coffe,burger;
12.    Button buttonOrder;
13.    @Override
14.    protected void onCreate(Bundle savedInstanceState) {
15.        super.onCreate(savedInstanceState);
16.        setContentView(R.layout.activity_main);
17.        addListenerOnButtonClick();
18.    }
19.    public void addListenerOnButtonClick(){
20.        //Getting instance of CheckBoxes and Button from the activty_main.
   xml file
21.        pizza=(CheckBox)findViewById(R.id.checkBox);
22.        coffe=(CheckBox)findViewById(R.id.checkBox2);
23.        burger=(CheckBox)findViewById(R.id.checkBox3);
24.        buttonOrder=(Button)findViewById(R.id.button);
25.
26.        //Applying the Listener on the Button click
27.        buttonOrder.setOnClickListener(new View.OnClickListener(){
28.
```

```
29.         @Override
30.         public void onClick(View view) {
31.             int totalamount=0;
32.             StringBuilder result=new StringBuilder();
33.             result.append("Selected Items:");
34.             if(pizza.isChecked()){
35.                 result.append("\nPizza 100Rs");
36.                 totalamount+=100;
37.             }
38.             if(coffe.isChecked()){
39.                 result.append("\nCoffe 50Rs");
40.                 totalamount+=50;
41.             }
42.             if(burger.isChecked()){
43.                 result.append("\nBurger 120Rs");
44.                 totalamount+=120;
45.             }
46.             result.append("\nTotal: "+totalamount+"Rs");
47.             //Displaying the message on the toast
48.             Toast.makeText(getApplicationContext(), result.toString(), Toast.
    LENGTH_LONG).show();
49.         }
50.
51.     });
52.  }
53.}
```

Output:



Android Custom CheckBox

Android provides facility to customize the UI of view elements rather than default.

You are able to create custom CheckBox in android. So, you can add some different images of checkbox on the layout.

Example of Custom CheckBox

In this example, we create both default as well as custom checkbox. Add the following code in activity_main.xml file.

activity_main.xml

File: activity_main.xml

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


    <TextView
        android:id="@+id/textView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
```

```
99.     android:textSize="25dp"
100.          android:text="Default Check Box"
101.          android:layout_alignParentTop="true"
102.          android:layout_alignParentLeft="true"
103.          android:layout_alignParentStart="true" />
104.
105.        <CheckBox
106.          android:layout_width="wrap_content"
107.          android:layout_height="wrap_content"
108.          android:text="New CheckBox"
109.          android:id="@+id/checkBox"
110.          android:layout_below="@+id/textView1"
111.          android:layout_centerHorizontal="true"
112.          android:layout_marginTop="46dp" />
113.
114.        <CheckBox
115.          android:layout_width="wrap_content"
116.          android:layout_height="wrap_content"
117.          android:text="New CheckBox"
118.          android:id="@+id/checkBox2"
119.          android:layout_below="@+id/checkBox"
120.          android:layout_alignLeft="@+id/checkBox"
121.          android:layout_alignStart="@+id/checkBox" />
122.
123.        <View
124.          android:layout_width="fill_parent"
125.          android:layout_height="1dp"
126.          android:layout_marginTop="200dp"
127.          android:background="#B8B894"
128.          android:id="@+id/viewStub" />
129.
130.        <CheckBox
131.          android:layout_width="wrap_content"
132.          android:layout_height="wrap_content"
133.          android:text="CheckBox 1"
134.          android:id="@+id/checkBox3"
135.          android:button="@drawable/customcheckbox"
136.          android:layout_below="@+id/viewStub"
137.          android:layout_centerHorizontal="true"
138.          android:layout_marginTop="58dp" />
139.
140.        <CheckBox
141.          android:layout_width="wrap_content"
142.          android:layout_height="wrap_content"
143.          android:text="CheckBox 2"
144.          android:id="@+id/checkBox4"
145.          android:button="@drawable/customcheckbox"
146.          android:layout_below="@+id/checkBox3"
147.          android:layout_alignLeft="@+id/checkBox3"
```

```
            android:layout_alignStart="@+id/checkBox3" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textSize="25dp"
            android:text="Custom Check Box"
            android:id="@+id/textView"
            android:layout_alignTop="@+id/viewStub"
            android:layout_centerHorizontal="true" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Show Checked"
            android:id="@+id/button"
            android:layout_alignParentBottom="true"
            android:layout_centerHorizontal="true" />

    </RelativeLayout>
```

Now implement a selector in another file (checkbox.xml) under drawable folder which customizes the checkbox.

checkbox.xml

File: checkbox.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_checked="true" android:drawable="@drawable/checked" />
    <item android:state_checked="false" android:drawable="@drawable/unchecked"/>
</selector>
```

Activity class

File: MainActivity.java

```
package example.javatpoint.com.customcheckbox;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    CheckBox cb1,cb2;
```

```
60.    Button button;
61.    @Override
62.    protected void onCreate(Bundle savedInstanceState) {
63.        super.onCreate(savedInstanceState);
64.        setContentView(R.layout.activity_main);
65.        cb1=(CheckBox)findViewById(R.id.checkBox3);
66.        cb2=(CheckBox)findViewById(R.id.checkBox4);
67.        button=(Button)findViewById(R.id.button);
68.
69.        button.setOnClickListener(new View.OnClickListener() {
70.            @Override
71.            public void onClick(View v) {
72.                StringBuilder sb=new StringBuilder("");
73.
74.                if(cb1.isChecked()){
75.                    String s1=cb1.getText().toString();
76.                    sb.append(s1);
77.                }
78.
79.                if(cb2.isChecked()){
80.                    String s2=cb2.getText().toString();
81.                    sb.append("\n"+s2);
82.
83.                }
84.                if(sb!=null && !sb.toString().equals("")){
85.                    Toast.makeText(getApplicationContext(), sb, Toast.LENGTH_LONG).show(
    );
86.
87.                }
88.                else{
89.                    Toast.makeText(getApplicationContext(),"Nothing Selected", Toast.LENGT
    H_LONG).show();
90.                }
91.
92.            }
93.
94.        });
95.    }
96. }
```

5.2 RadioButton

5.2. 1. Event methods of RadioButton
5.2.2. Dynamic and Custom RadioButton

Android RadioButton

RadioButton is a two states button which is either checked or unchecked.

If a single radio button is unchecked, we can click it to make checked radio button. Once a radio button is checked, it cannot be marked as unchecked by user.

RadioButton is generally used with *RadioGroup*. RadioGroup contains several radio buttons, marking one radio button as checked makes all other radio buttons as unchecked.

Example of Radio Button:

In this example, we are going to implement single radio button separately as well as radio button in RadioGroup.

activity_main.xml

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    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"
    tools:context="example.javatpoint.com.radiobutton.MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:gravity="center_horizontal"
        android:textSize="22dp"
        android:text="Single Radio Buttons" />



    <!--  Default RadioButtons  -->

    <RadioButton
        android:id="@+id/radioButton1"
        android:layout_width="fill_parent"
```

```
26.     android:layout_height="wrap_content"
27.     android:layout_gravity="center_horizontal"
28.     android:text="Radio Button 1"
29.     android:layout_marginTop="20dp"
30.
31.     android:textSize="20dp" />
32.   <RadioButton
33.     android:id="@+id/radioButton2"
34.     android:layout_width="fill_parent"
35.     android:layout_height="wrap_content"
36.     android:text="Radio Button 2"
37.     android:layout_marginTop="10dp"
38.
39.     android:textSize="20dp" />
40.
41.
42.   <View
43.     android:layout_width="fill_parent"
44.     android:layout_height="1dp"
45.     android:layout_marginTop="20dp"
46.     android:background="#B8B894" />
47.
48.   <TextView
49.     android:id="@+id/textView2"
50.     android:layout_width="fill_parent"
51.     android:layout_height="wrap_content"
52.     android:layout_marginTop="30dp"
53.     android:gravity="center_horizontal"
54.     android:textSize="22dp"
55.     android:text="Radio button inside RadioGroup" />
56.
57.
58.   <!--   Customized RadioButtons  -->
59.
60.
61.   <RadioGroup
62.     android:layout_width="wrap_content"
63.     android:layout_height="wrap_content"
64.     android:id="@+id/radioGroup">
65.
```

```
66.      <RadioButton
67.         android:id="@+id/radioMale"
68.         android:layout_width="fill_parent"
69.         android:layout_height="wrap_content"
70.         android:text="  Male"
71.         android:layout_marginTop="10dp"
72.         android:checked="false"
73.         android:textSize="20dp" />
74.
75.      <RadioButton
76.         android:id="@+id/radioFemale"
77.         android:layout_width="fill_parent"
78.         android:layout_height="wrap_content"
79.         android:text="   Female"
80.         android:layout_marginTop="20dp"
81.         android:checked="false"
82.
83.         android:textSize="20dp" />
84.   </RadioGroup>
85.
86.   <Button
87.      android:layout_width="wrap_content"
88.      android:layout_height="wrap_content"
89.      android:text="Show Selected"
90.      android:id="@+id/button"
91.      android:onClick="onclickbuttonMethod"
92.      android:layout_gravity="center_horizontal" />
93.
94.
95.</LinearLayout>
```

Activity class

File: MainActivity.java

```
1.  package example.javatpoint.com.radiobutton;
2.
3.  import android.support.v7.app.AppCompatActivity;
4.  import android.os.Bundle;
5.  import android.view.View;
6.  import android.widget.Button;
7.  import android.widget.RadioButton;
```

```
8.  import android.widget.RadioGroup;
9.  import android.widget.Toast;
10.
11.public class MainActivity extends AppCompatActivity {
12.    Button button;
13.    RadioButton genderradioButton;
14.    RadioGroup radioGroup;
15.    @Override
16.    protected void onCreate(Bundle savedInstanceState) {
17.        super.onCreate(savedInstanceState);
18.        setContentView(R.layout.activity_main);
19.        radioGroup=(RadioGroup)findViewById(R.id.radioGroup);
20.    }
21.    public void onclickbuttonMethod(View v){
22.        int selectedId = radioGroup.getCheckedRadioButtonId();
23.        genderradioButton = (RadioButton) findViewById(selectedId);
24.        if(selectedId==-1){
25.            Toast.makeText(MainActivity.this,"Nothing selected", Toast.LENGT
    H_SHORT).show();
26.        }
27.        else{
28.            Toast.makeText(MainActivity.this,genderradioButton.getText(), Toa
    st.LENGTH_SHORT).show();
29.        }
30.
31.    }
32.}
```

Android Dynamic RadioButton

Instead of creating RadioButton through drag and drop from palette, android also facilitates you to create it programmatically (dynamically).

For creating dynamic RadioButton,
we need to use **android.view.ViewGroup.LayoutParams** which configures the width and height of views and
implements *setOnCheckedChangeListener()* method of *RadioGroup* class.

Example of Dynamic RadioButton

activity_main.xml

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:id="@+id/relativeLayout"
    tools:context="com.example.test.dynamicradiobutton.MainActivity">

</RelativeLayout>
```

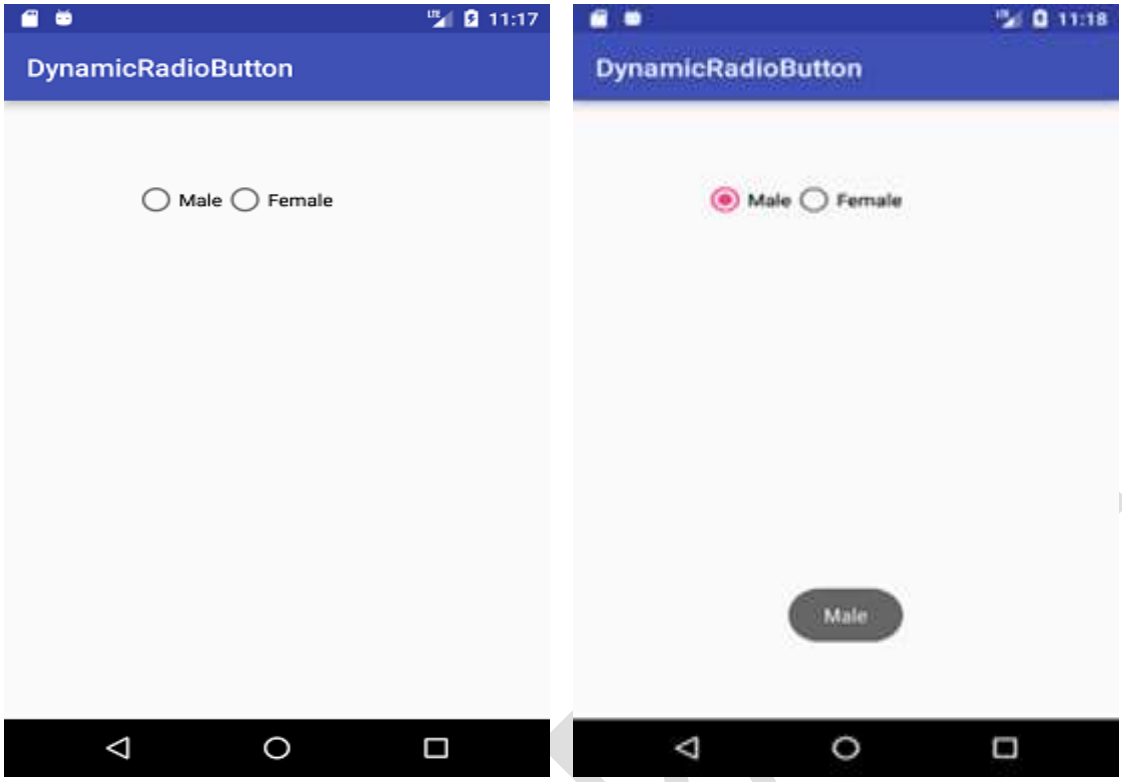
Activity class

File: MainActivity.java

```
package com.example.test.dynamicradiobutton;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.RelativeLayout;

import android.widget.RelativeLayout.LayoutParams;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    RadioGroup rg;
    RelativeLayout rl;
    RadioButton rb1,rb2;

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

```

```
22.       rg = new RadioGroup(this);
23.       rl = (RelativeLayout) findViewById(R.id.relativeLayout);
24.       rb1 = new RadioButton(this);
25.       rb2 = new RadioButton(this);
26.
27.       rb1.setText("Male");
28.       rb2.setText("Female");
29.       rg.addView(rb1);
30.       rg.addView(rb2);
31.       rg.setOrientation(RadioGroup.HORIZONTAL);
32.
33.       RelativeLayout.LayoutParams params = new RelativeLayout.LayoutP
    arams((int) LayoutParams.WRAP_CONTENT,(int)LayoutParams.WRAP_CO
    NTENT);
34.       params.leftMargin =150;
35.       params.topMargin = 100;
36.
37.       rg.setLayoutParams(params);
38.       rl.addView(rg);
39.
40.       rg.setOnCheckedChangeListener(new RadioGroup.OnCheckedChang
    eListener() {
41.           @Override
42.           public void onCheckedChanged(RadioGroup group, int checkedId)
     {
43.               RadioButton radioButton = (RadioButton) findViewById(checked
    Id);
44.               Toast.makeText(getApplicationContext(),radioButton.getText(),T
    oast.LENGTH_LONG).show();
45.           }
46.       });
47.   }
48.}
```

Output:



5.3 Spinner , AlertDialog

Spinner

Android Spinner is like the combox box of AWT or Swing.

It can be used to display the multiple options to the user in which only one item can be selected by the user.

Android spinner is like the drop down menu with multiple values from which the end user can select only one value.

Android spinner is associated with AdapterView. So you need to use one of the adapter classes with spinner.

Android Spinner class is the subclass of AsbSpinner class.

Android Spinner Example

In this example, we are going to display the country list. You need to use ArrayAdapter class to store the country list.

activity_main.xml

Drag the Spinner from the pallete, now the activity_main.xml file will like this:

File: activity_main.xml

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

    <Spinner
        android:id="@+id/spinner"
        android:layout_width="149dp"
        android:layout_height="40dp"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.502"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.498" />

```

```
24.</android.support.constraint.ConstraintLayout>
```

Activity class

Let's write the code to display item on the spinner and perform event handling.

File: MainActivity.java

```
1.  package example.javatpoint.com.spinner;
2.
3.  import android.support.v7.app.AppCompatActivity;
4.  import android.os.Bundle;
5.  import android.view.View;
6.  import android.widget.AdapterView;
7.  import android.widget.ArrayAdapter;
8.  import android.widget.Spinner;
9.  import android.widget.Toast;
10.
11.public class MainActivity extends AppCompatActivity implements
12.        AdapterView.OnItemSelectedListener {
13.    String[] country = { "India", "USA", "China", "Japan", "Other"};
14.
15.    @Override
16.    protected void onCreate(Bundle savedInstanceState) {
17.        super.onCreate(savedInstanceState);
18.        setContentView(R.layout.activity_main);
19.       //Getting the instance of Spinner and applying OnItemSelectedListen
     er on it
20.        Spinner spin = (Spinner) findViewById(R.id.spinner);
21.        spin.setOnItemSelectedListener(this);
```

```
22.
23.       //Creating the ArrayAdapter instance having the country list
24.       ArrayAdapter aa = new ArrayAdapter(this,android.R.layout.simple_s
      pinner_item,country);
25.       aa.setDropDownViewResource(android.R.layout.simple_spinner_dr
      opdown_item);
26.       //Setting the ArrayAdapter data on the Spinner
27.       spin.setAdapter(aa);
28.
29.    }
30.
31.    //Performing action onItemSelected and onNothing selected
32.    @Override
33.    public void onItemSelected(AdapterView<?> arg0, View arg1, int positi
      on, long id) {
34.       Toast.makeText(getApplicationContext(),country[position] , Toast.LE
      NGTH_LONG).show();
35.    }
36.    @Override
37.    public void onNothingSelected(AdapterView<?> arg0) {
38.       // TODO Auto-generated method stub
39.    }
40.}
```


AlertDialog

Android AlertDialog can be used to display the dialog message with OK and Cancel buttons.

It can be used to interrupt and ask the user about his/her choice to continue or discontinue.

Android AlertDialog is composed of three regions: title, content area and action buttons.

Android AlertDialog is the subclass of Dialog class.

Methods of AlertDialog class

Method	Description
public AlertDialog.Builder setTitle(CharSequence)	This method is used to set the title of AlertDialog.
public AlertDialog.Builder setMessage(CharSequence)	This method is used to set the message fo AlertDialog.
public AlertDialog.Builder setIcon(int)	This method is used to set the icon over AlertDialog.

Android AlertDialog Example

activity_main.xml

You can have multiple components, here we are having only a textview.

File: activity_main.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.constraint.ConstraintLayout xmlns:android="http://sch
   emas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     xmlns:tools="http://schemas.android.com/tools"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent"
7.     tools:context="example.javatpoint.com.alertdialog.MainActivity">
```

```
8.
9.     <Button
10.        android:layout_width="wrap_content"
11.        android:layout_height="wrap_content"
12.        android:id="@+id/button"
13.        android:text="Close app"
14.        app:layout_constraintBottom_toBottomOf="parent"
15.        app:layout_constraintLeft_toLeftOf="parent"
16.        app:layout_constraintRight_toRightOf="parent"
17.        app:layout_constraintTop_toTopOf="parent" />
18.
19.</android.support.constraint.ConstraintLayout>
```

strings.xml

Optionally, you can store the dialog message and title in the strings.xml file.

File: strings.xml

```
1. <resources>
2.     <string name="app_name">AlertDialog</string>
3.     <string name="dialog_message">Welcome to Alert Dialog</string>
4.     <string name="dialog_title">Javatpoint Alert Dialog</string>
5. </resources>
```

Activity class

File: MainActivity.java

```
1. package example.javatpoint.com.alertdialog;
2.
```

```
3. import android.content.DialogInterface;
4. import android.support.v7.app.AppCompatActivity;
5. import android.os.Bundle;
6. import android.view.View;
7. import android.widget.Button;
8. import android.app.AlertDialog;
9. import android.widget.Toast;
10.
11.public class MainActivity extends AppCompatActivity {
12.    Button closeButton;
13.    AlertDialog.Builder builder;
14.    @Override
15.    protected void onCreate(Bundle savedInstanceState) {
16.        super.onCreate(savedInstanceState);
17.        setContentView(R.layout.activity_main);
18.
19.        closeButton = (Button) findViewById(R.id.button);
20.        builder = new AlertDialog.Builder(this);
21.        closeButton.setOnClickListener(new View.OnClickListener() {
22.            @Override
23.            public void onClick(View v) {
24.
25.                //Uncomment the below code to Set the message and title from
    the strings.xml file
26.                builder.setMessage(R.string.dialog_message) .setTitle(R.string.d
    ialog_title);
27.
```

```
28.         //Setting message manually and performing action on button cli
    ck
29.         builder.setMessage("Do you want to close this application ?")
30.                 .setCancelable(false)
31.                 .setPositiveButton("Yes", new DialogInterface.OnClickListen
    er() {
32.                     public void onClick(DialogInterface dialog, int id) {
33.                         finish();
34.                         Toast.makeText(getApplicationContext(),"you choose y
    es action for alertbox",
35.                         Toast.LENGTH_SHORT).show();
36.                     }
37.                 })
38.                 .setNegativeButton("No", new DialogInterface.OnClickListe
    ner() {
39.                     public void onClick(DialogInterface dialog, int id) {
40.                         //  Action for 'NO' Button
41.                         dialog.cancel();
42.                         Toast.makeText(getApplicationContext(),"you choose n
    o action for alertbox",
43.                         Toast.LENGTH_SHORT).show();
44.                     }
45.                 });
46.         //Creating dialog box
47.         AlertDialog alert = builder.create();
48.         //Setting the title manually
49.         alert.setTitle("AlertDialogExample");
50.         alert.show();
```

```
51.          }
52.       });
53.    }
54.}
```

5.4 AutocompleteTextview, Textwatcher to EditText

AutocompleteTextview

Android AutoCompleteTextView completes the word based on the reserved words, so no need to write all the characters of the word.

Android AutoCompleteTextView is a editable text field, it displays a list of suggestions in a drop down menu from which user can select only one suggestion or value.

Android AutoCompleteTextView is the subclass of EditText class. The MultiAutoCompleteTextView is the subclass of AutoCompleteTextView class.

Android AutoCompleteTextView Example

In this example, we are displaying the programming languages in the autocompletetextview. All the programming languages are stored in string array. We are using the ArrayAdapter class to display the array content.

Let's see the simple example of autocompletetextview in android.

activity_main.xml

Drag the AutoCompleteTextView and TextView from the pallete, now the activity_main.xml file will like this:

File: activity_main.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.constraint.ConstraintLayout xmlns:android="http://sch
   emas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     xmlns:tools="http://schemas.android.com/tools"
```

```
5.    android:layout_width="match_parent"
6.    android:layout_height="match_parent"
7.    tools:context="example.javatpoint.com.autocompletetextview.MainAc
    tivity">
8.
9.    <TextView
10.       android:layout_width="wrap_content"
11.       android:layout_height="wrap_content"
12.       android:text="What is your favourite programming language?"
13.       app:layout_constraintBottom_toBottomOf="parent"
14.       app:layout_constraintLeft_toLeftOf="parent"
15.       app:layout_constraintRight_toRightOf="parent"
16.       app:layout_constraintTop_toTopOf="parent"
17.       app:layout_constraintVertical_bias="0.032" />
18.
19.   <AutoCompleteTextView
20.       android:id="@+id/autoCompleteTextView"
21.       android:layout_width="200dp"
22.       android:layout_height="wrap_content"
23.       android:layout_marginLeft="92dp"
24.       android:layout_marginTop="144dp"
25.       android:text=""
26.       app:layout_constraintStart_toStartOf="parent"
27.       app:layout_constraintTop_toTopOf="parent" />
28.
29.</android.support.constraint.ConstraintLayout>
```

Activity class

Let's write the code of AutoCompleteTextView.

File: MainActivity.java

```
1. package example.javatpoint.com.autocompletetextview;
2.
3. import android.graphics.Color;
4. import android.support.v7.app.AppCompatActivity;
5. import android.os.Bundle;
6. import android.widget.ArrayAdapter;
7. import android.widget.AutoCompleteTextView;
8.
9. public class MainActivity extends AppCompatActivity {
10.    String[] language ={"C","C++","Java",".NET","iPhone","Android","ASP.NET","PHP"};
11.    @Override
12.    protected void onCreate(Bundle savedInstanceState) {
13.        super.onCreate(savedInstanceState);
14.        setContentView(R.layout.activity_main);
15.        //Creating the instance of ArrayAdapter containing list of language names
16.        ArrayAdapter<String> adapter = new ArrayAdapter<String>
17.                (this,android.R.layout.select_dialog_item,language);
18.        //Getting the instance of AutoCompleteTextView
19.        AutoCompleteTextView actv =  (AutoCompleteTextView)findViewById(R.id.autoCompleteTextView);
20.        actv.setThreshold(1);//will start working from first character
```

```
21.       actv.setAdapter(adapter);//setting the adapter data into the AutoCo
      mpleteTextView
22.       actv.setTextColor(Color.RED);
23.    }
24.}
```

EditText to Textwatcher

Android EditText with TextWatcher (Searching data from ListView)

Android **EditText** is a subclass of *TextView*. EditText is used for entering and modifying text.

While using EditText width, we must specify its input type in *inputType* property of EditText which configures the keyboard according to input.

EditText uses **TextWatcher** interface to watch change made over EditText. For doing this, EditText calls the *addTextChangedListener()* method.

Methods of TextWatcher:

1. **beforeTextChanged(CharSequence arg0, int arg1, int arg2, int arg3):** It is executed before making any change over EditText.
2. **onTextChanged(CharSequence cs, int arg1, int arg2, int arg3):** It is executed while making any change over EditText.
3. **afterTextChanged(Editable arg0):** It is executed after change made over EditText.

Example of EditText with TextWatcher()

In this example, we will implement EditText with TextWatcher to search data from ListView.

activity_main.xml

Create an activity_main.xml file in layout folder containing EditText and ListView.

File: activity_main.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android
   .com/apk/res/android"
3.     xmlns:tools="http://schemas.android.com/tools"
4.     android:layout_width="match_parent"
5.     android:layout_height="match_parent"
6.     android:paddingBottom="@dimen/activity_vertical_m
   argin"
7.     android:paddingLeft="@dimen/activity_horizontal_m
   argin"
8.     android:paddingRight="@dimen/activity_horizontal_
   margin"
9.     android:paddingTop="@dimen/activity_vertical_margi
   n"
10.          tools:context="com.example.test.searchfromlistvi
   ew.MainActivity">
11.
```

```
12.     <EditText
13.         android:layout_width="wrap_content"
14.         android:layout_height="wrap_content"
15.         android:id="@+id/editText"
16.         android:inputType="text"
17.         android:layout_alignParentTop="true"
18.         android:layout_alignParentLeft="true"
19.         android:layout_alignParentStart="true"
20.         android:layout_alignParentRight="true"
21.         android:layout_alignParentEnd="true" />
22.
23.     <ListView
24.         android:layout_width="wrap_content"
25.         android:layout_height="wrap_content"
26.         android:id="@+id/listView"
27.         android:layout_below="@+id/editText"
28.         android:layout_alignParentLeft="true"
29.         android:layout_alignParentStart="true" />
30. </RelativeLayout>
```

Create another file list_item.xml in layout folder which contains data of ListView.

list_item.xm

File: list_item.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout
3.     xmlns:android="http://schemas.android.com/apk/res/
   android"
4.     android:orientation="vertical"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent">
7.
8. <TextView android:id="@+id/product_name"
9.     android:layout_width="fill_parent"
10.         android:layout_height="wrap_content"
11.         android:padding="10dip"
12.         android:textSize="16dip"
13.         android:textStyle="bold"/>
14.     </LinearLayout>
```

Activity class

Activity class

```
1. package com.example.test.searchfromlistview;
2.
3. import android.os.Bundle;
4. import android.text.Editable;
```

```
5. import android.text.TextWatcher;
6. import android.widget.ArrayAdapter;
7. import android.widget.EditText;
8. import android.widget.ListView;
9. import android.support.v7.app.AppCompatActivity;
10.     import android.widget.Toast;
11.
12.     public class MainActivity extends AppCompatActivi
    ty {
13.
14.         private ListView lv;
15.         private EditText editText;
16.         private ArrayAdapter<String> adapter;
17.
18.         private String products[] = {"Apple", "Banana","Pi
    napple", "Orange", "Papaya", "Melon",
19.                 "Grapes", "Water Melon","Lychee", "Guava",
     "Mango", "Kivi"};
20.         @Override
21.         protected void onCreate(Bundle savedInstanceSt
    ate) {
22.             super.onCreate(savedInstanceState);
23.             setContentView(R.layout.activity_main);
```

```
24.
25.             lv = (ListView) findViewById(R.id.listView);
26.             editText = (EditText) findViewById(R.id.editText)
    ;
27.             adapter = new ArrayAdapter<String>(this, R.lay
    out.list_item, R.id.product_name, products);
28.             lv.setAdapter(adapter);
29.
30.             editText.addTextChangedListener(new TextWat
    cher() {
31.
32.                 @Override
33.                 public void onTextChanged(CharSequence cs
    , int arg1, int arg2, int arg3) {
34.                     adapter.getFilter().filter(cs);
35.                 }
36.
37.                 @Override
38.                 public void beforeTextChanged(CharSequenc
    e arg0, int arg1, int arg2, int arg3) {
39.                     Toast.makeText(getApplicationContext(),"b
    efore text change",Toast.LENGTH_LONG).show();
40.                 }
```

```
41.
42.                 @Override
43.                 public void afterTextChanged(Editable arg0) {

44.                     Toast.makeText(getApplicationContext(),"a
    fter text change",Toast.LENGTH_LONG).show();
45.                 }
46.             });
47.         }
48.     }
```

Output: