UNIT-2

BASIC WIDGETS

Understanding the Role of Android Application Components:

To understand the role and functionality of different components that make up an Android application, you want to create a new Android application and study each of its components.

Let's start by creating an application that prompts the user to enter a name and displays a **Welcome message** in return.

To create an application, open Eclipse and choose, File→ New→ Android Application Project or click the Android Project Creator icon on the Eclipse toolbar. You see a dialog box, as shown in Figure 2.1 (left).

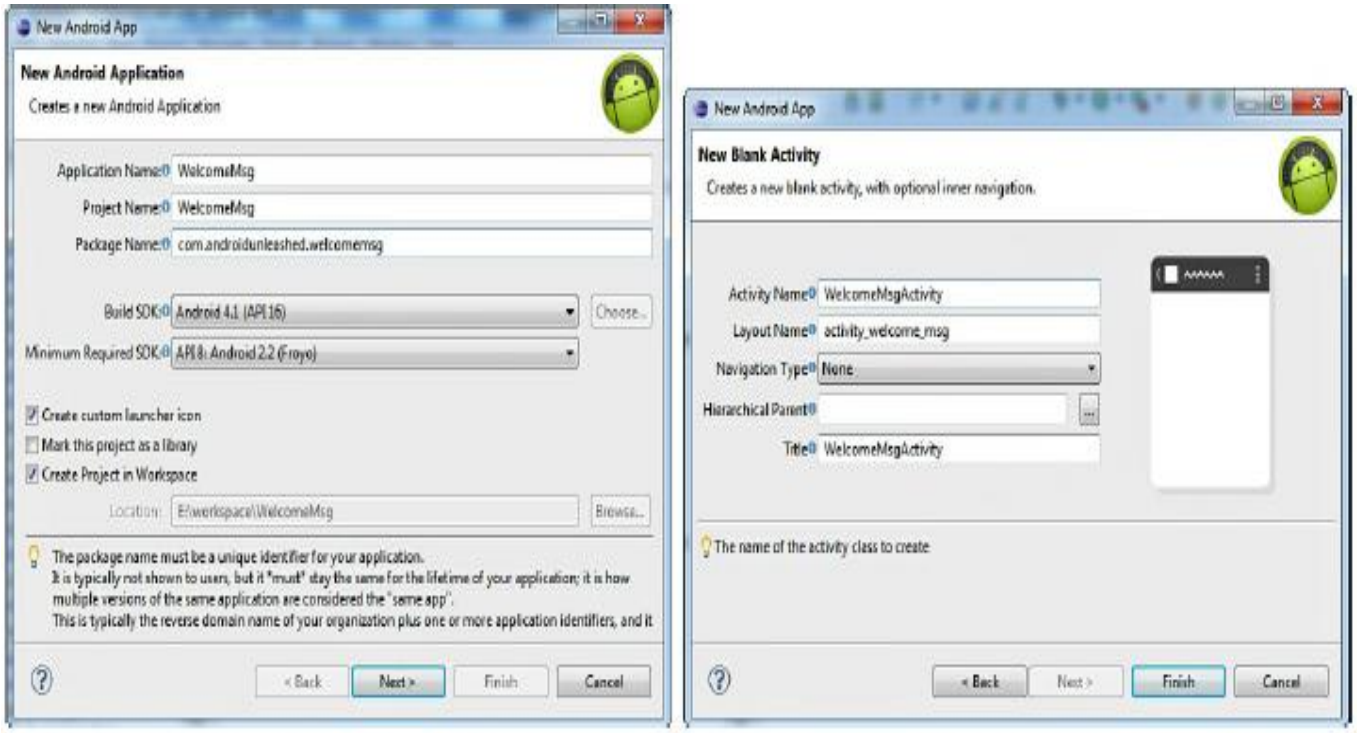


Figure 2.1. Specifying the project name (left), and specifying the application target (right)

- In the **Application Name** box, enter the name of the Android project as WelcomeMsg.
- Then the **Project Name** is automatically assigned, which is the same as the application name by default.
- Being a unique identifier, the **Package Name** is com.androidunleashed.welcomemsg.
- From the **Build SDK** drop-down, select Android 4.1 (API 16) as the target platform.

- select the **Create Project in Workspace** check box to create the project files at the Workspace location.
- The **Create custom launcher icon** check box is selected by default and enables us to define the icon of our application.
- Click the **Next** button to move to the next dialog box.
- The next dialog prompts us to select whether we want to create an activity, select the Create Activity check box and the **Blank Activity** option from the list and then click Next.
- The next dialog (see Figure 2.1—right) asks us to enter information about the newly created activity. Name the activity **WelcomeMsgActivity.** The layout filename and title name automatically change to reflect the newly assigned activity name. The layout name becomes **activity_welcome_msg,** and the Title of the application also changes to **WelcomeMsgActivity.**
- click the **Finish** button after supplying the required information.

Application Components:

App components are essential ones which specify the building blocks of an application. Each and every component usually acts as an entry point to application. The components may interlink i.e, each component may depend on others.

Basic components of Android are:-

- Activities
- Services
- Broadcast Recievers
- Content Providers.

Activity: An activity represents a single screen with a user interface,in-short Activity performs actions on the screen. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

An activity is implemented as a subclass of **Activity** class as follows –

```
public class MainActivity extends Activity {

}
```

Services: A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

A service is implemented as a subclass of **Service** class as follows –

```
public class MyService extends Service {

}
```

Broadcast Receivers: Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

A broadcast receiver is implemented as a subclass of **BroadcastReceiver**class and each message is broadcaster as an **Intent** object.

```
public class MyReceiver  extends  BroadcastReceiver {

  public void onReceive(context,intent){}

}
```

Content Providers: A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver*class. The data may be stored in the file system, the database or somewhere else entirely.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends  ContentProvider {

  public void onCreate(){}
```

Understanding the Utility of Android API:

- API Level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform.
- The initial release of the Android platform provided API Level 1, and subsequent releases have incremented the API level.
- The Android platform provides a framework API that applications can use to interact with the underlying Android system. The framework API consists of:

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

Table 2.1. The Android Platform and Corresponding API Levels

Platform Version	API Level	Code Name
Android 4.1	16	Jelly Bean
Android 4.0.3	15	Ice Cream Sandwich
Android 4.0	14	Ice Cream Sandwich
Android 3.2	13	Honeycomb
Android 3.1	12	Honeycomb
Android 3.0	11	Honeycomb
Android 2.3.3	10	Gingerbread
Android 2.3.1	9	Gingerbread
Android 2.2	8	Froyo
Android 2.1	7	Eclair
Android 2.0.1	6	Eclair
Android 2.0	5	Eclair
Android 1.6	4	Donut
Android 1.5	3	Cupcake
Android 1.1	2	
Android 1.0	1	

Overview of the Android Project Files:

The following files and directories are created for the Android application.

/src folder—This folder contains the entire Java source file of the application and a directory structure corresponding to the package name supplied in the application which contains the project's default package: com.androidunleashed.welcomemsg. On expanding the package, you find the Activity of the application, the WelcomeMsgActivity.java file, within it.

/src/com.androidunleashed.welcomemsg—Refers to the package name of theapplication.

/src/com.androidunleashed.welcomemsg/WelcomeMsgActivity.java—This is the default Activity file of the application. Recall that each application has at least one Activity that acts as the main entry point to the application. The Activity file is automatically defined as the default launch Activity in the Android Manifest file.

/gen folder—Contains Java files generated by ADT on compiling the application and R.java file that contains references for all the resources defined in the res directory. It also contains a BuildConfig.java file that is used to run code only in debug mode.

/gen/com.androidunleashed.welcomemsg/R.java—All the layout and other resource information that is coded in the XML files is converted into Java source code and placed in the R.java file. It also means that the file contains the ID of all the resources of the application. The R.java file is compiled into the Java byte code files and then converted into .dex format.

Android SDK jar file—The jar file for the target platform.

/assets folder—The assets folder is empty by default. It stores raw asset files that may be required in the application. It may contain fonts, external JAR files, and so on to be used in the application.

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

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

/res/drawable-xhdpi, /res/drawable-hdpi, /res/drawable-mdpi, /res/drawable-ldpi—the application's icon and graphic resources are kept in these folders. Usually, graphics of 320dpi,

240dpi, 160dpi, and 120dpi are used in Android applications. The graphics with 320dpi, 240dpi, 160dpi, and 120dpi are stored in the res/drawable-xhdpi,res/drawable-hdpi/, res/drawable-mdpi, and res/drawable-ldpi folders, respectively.

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

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

/res/layout/activity_welcome_msg.xml—The layout file used by WelcomeMsgActivity to draw views on the screen. The views or controls are laid in the specified layout.

/res/values/strings.xml—Contains the string resources. String resources contain the text matter to be assigned to different controls of the applications. This file also defines string arrays.

AndroidManifest.xml—The central configuration file for the application.

proguard.cfg—Defines how ProGuard optimizes the application's code. ProGuard is a tool that removes unused code from the Android application and optimizes it, which increases performance.

project.properties—

A build file used by Eclipse and the Android ADT plug-in.

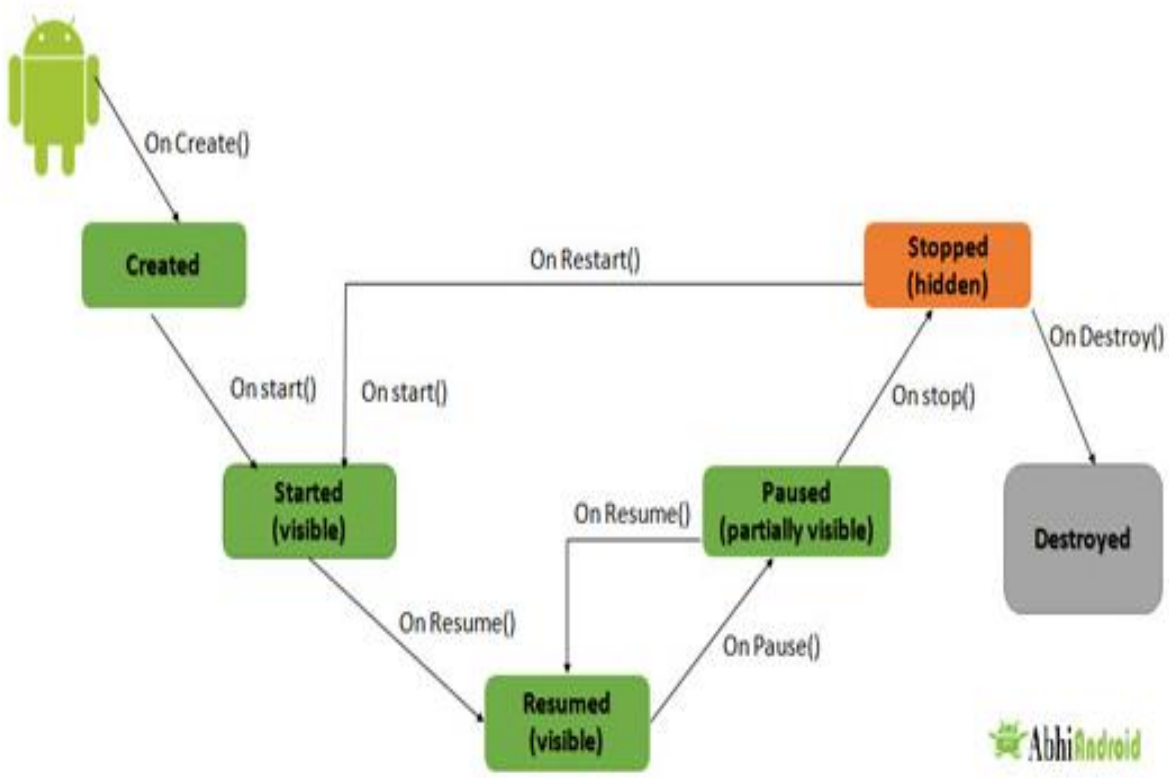
- It contains project settings such as the build target. You can use this file to change variousproperties of the project.
- Other folders that may become the part of an Android application include bin, libs, and referenced libraries.
- The bin folder is hidden.
- The libs and referenced libraries folders don't appear until a third-party library and reference are added to the project.

Understanding Activities:

- Every unique screen the user interacts with in an application is displayed through an Activity.
- There is one Activity for each screen.
- A simple application may consist of just one Activity, whereas large applications contain several Activities.
- Each Activity of an application operates independently of the others.

- A stack of Activities is maintained while running an application, and the Activity at the top of the stack is the one currently being displayed. When the Back button is pressed, the Activity is popped from the stack, making the previous Activity the current Activity, which therefore displays the previous screen.
- The transition from one Activity to another is accomplished through the use of asynchronous messages called **intents**. where these Intents can be used to pass data from one Activity to another.
- All of the Activities in the application must be defined in the Application's manifest file.
- Each Activity in an Android application is either a direct subclass of the Activity base class or a subclass of an Activity subclass.
- Activities have life cycles and inherit a set of life cycle methods from the Activity base class that are executed when the corresponding life cycle state of the Activity is reached**.**

The Android Activity Life Cycle



- **onCreate()** – Called when the activity is first created

- **onStart()** – Called just after it's creation or by restart method after onStop(). Here Activity start becoming visible to user
- **onResume()** – Called when Activity is visible to user and user can interact with it
- **onPause()** – Called when Activity content is not visible because user resume previous activity
- **onStop()** – Called when activity is not visible to user because some other activity takes place of it
- **onRestart()** – Called when user comes on screen or resume the activity which was stopped
- **onDestroy** – Called when Activity is not in background

Example program of the Android Lifecycle

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast.makeText(this, "on start",
Toast.LENGTH_SHORT).show();

    }
    @Override
    protected void onResume()
    {
        super.onResume();
        Toast.makeText(this, "on Resume",
Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onStart() {
        super.onStart();
        Toast.makeText(this, "on start",
Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onStop() {
        super.onStop();
```

```
        Toast.makeText(this, "on Stop",
Toast.LENGTH_SHORT).show();

    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, " on destroy",
Toast.LENGTH_SHORT).show();
    }
}
```

Role of the Android Manifest File:

- Manifest file is kept in Project's root directory.
- Manifest is an XML file that defines the overall structure and information about the application.
- To run the applications we need to have permissions for activities, services and intents.
- The manifest also defines metadata of the application such as its icon and label.

```
<manifest

xmlns:android="http://schemas.android.com/apk/res/android"

package="com.androidunleashed.welcomemsg"

android:versionCode="1"

android:versionName="1.0" >

<uses-sdk

android:minSdkVersion="8"

android:targetSdkVersion="15" />

   <application

            android:icon="@drawable/ic_launcher"

             android:label="@string/app_name"

             android:theme="@style/AppTheme" >

<activity

              android:name=".WelcomeMsgActivity"
```

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

- **android**—Identifies the Android namespace used to provide several system attributes used within the application.
- **package**—Its value is set to the application's Java package.
- **Version code-**is used to define the current application version.
- **Version name-**is used to specify a version number that is displayed to users.
- **<uses-sdk>—**This tag is optional and is used to specify the maximum, minimum, and preferred API level required for the application to operate.
- **android:minSdkVersion**—Used to specify the minimum API level required for this application to run. The default value is "1."
- **android:targetSdkVersion**—Used to specify the preferred API level on which the application is designed to run.
- **android:maxSdkVersion**—Used to specify the maximum API level supportable by the application
- **<application> tag**—Nested within <manifest> is <application>, which is the parent of application control tags
- **<activity> tag**—Nested within <application> is the <activity> tag, which describes an Activity component.
- **<intent- filter>—**Nested within <activity> is the <intent-filter>. The intents are used to interact with the applications and services.

- **<service> tags—**Used for declaring services. Services refer to the processes that run in the background without a user interface.
- **<receiver> tags—**Used for declaring broadcast receivers. Broadcast receivers are used to listen and respond to broadcast announcements.
- **lication> tag**—Nested within <manifest> is <application>, which is the parent of application control tags
- **<provider> tags**—Used for declaring content providers. Content providers provide content, that is, data to applications.
- **<uses-permission>** tags—Used for declaring the permissions that the application needs.
  Example:
  <uses-permission android:name="android.permission.CAMERA" />—Used for the application that needs to use the camera.
  <uses-permission android:name="android.permission.INTERNET"/>—Used for the application that needs to access the Internet

Creating the User Interface:

- You can create user interfaces entirely in Java code or entirely in XML or in both.
- All the user interface elements are made up of two things:
  1. View
  2. View Group
- View is the base class for widgets to create components such as Buttons, Text view, Edit text, checkbox, Radio button.
- View Group is a sub class of view.
- The XML file in which you create the user interface is activity_welcome_msg.xml found in the res/layout folder.
- In the activity_welcome_msg.xml file,different controls or views that you want to interact with the user are defined.
- The default layout in which the controls or views are usually arranged is RelativeLayout.
- Default Code in the activity_welcome_msg.xml File

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/
                    android”
xmlns:tools="http://schemas.android.com/tools"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent" >
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:layout_centerVertical="true"
android:text="@string/hello_world"
tools:context=".WelcomeMsgActivity" />
</RelativeLayout>
```

Commonly Used Layouts and Controls:

- The views or the controls that we want to display in an application are arranged in an order or sequence by placing them in the desired layout.
- The layouts also known as Containers or ViewGroups are used for organizing the views or controls in the required format.
- Types of Layouts:
    1. Linear layout
    2. Relative layout
    3. Absolute layout
    4. Frame layout
    5. Table layout
    6. Grid layout
    7. List View
    8. Scroll View
- **Linear layout: It** is a ViewGroup that aligns all children in a single direction, vertically or horizontally.
- **RelativeLayout :** It is a ViewGroup that displays child views in relative positions.
- **AbsoluteLayout :** It allows us to specify the exact location of the child views and widgets.
- **TableLayout** : It is a view that groups its child views into rows and columns
- **FrameLayout :** It is a placeholder on screen that is used to display a single view.
- **ScrollView :** It is a special type of FrameLayout in that it allows users to scroll through a list of views that occupy more space than the physical display. The ScrollView can contain only one child view or ViewGroup, which normally is a LinearLayout

- **ListView**: is a view group that displays a list of scrollable item
- **GridView:** is a ViewGroup that displays items in two-dimensional scrolling grid. The items in the grid come from the ListAdapter associated with this view

Controls:

- **TextView—**A read-only text label. It supports multiline display, string formatting, and automatic word wrapping.
- **EditText—**An editable text box that also accepts multiline entry and word-wrapping.
- **ListView—**A ViewGroup that creates and manages a vertical list of views, displaying them as rows within the list**.**
- **Spinner—**A TextView and an associated list of items that allows us to select an item from the list to display in the text box.
- **Button—**A standard command button.
- **CheckBox—**A button allowing a user to select (check) or unselect (uncheck).
- **RadioButton—**A mutually exclusive button, which, when selected, unselects all other.

Event Handling:

- **Event** is an action. It occurs when a user interact with mobile application. Events in Android are in many different forms like keystrokes, touch inputs and many others. Touch screen interaction falls in the category of touch events.
- Events:
  1. Event Listener
  2. Event Listener Registration
  3. Event Handlers
- **Event Listeners Registration** − Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.
- **Event Handlers** − When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.
- **Event Listener**

  It is an interface which contains a call back method. When user will trigger the view elements it will call these methods. Here are few important call back methods:

- **OnClickListener():** This event listener is called when a user clicks on any UI element like button, text or image. OnClick() event handler is used to handle this listener.
- **OnLongClickListener():** This method is called when a user clicks on any UI element for a long time or hold a UI element for few seconds. OnLongClick() event handler is used to handle this listener.
- **OnFocusChangeListener():** This method is called when a UI element or a widget losses its focus. Simply user navigates forward. OnFocusChange() event handler is used to handle this listener.
- **OnKeyListener():** This method is called when a user presses a key on keyboard. OnKey() event handler is used to handle this event.
- **OnTouchListener():** This method is called when a user touches any UI element on the screen like press or release a button. OnTouch() event handler is used to handle this listener.
- **OnMenuItemClickListener():** This method is used when a user clicks or selects a menu item.OnMenuItemClick() event handler is used to handle this listener.
- **OnCreateContextMenuListener():** OnCreateContextMenu() event handler is used to handle this listener.
- There are three ways of event handling:
  - Creating an anonymous inner class
  - Implementing the OnClickListener interface
  - Declaring the event handler in the XML definition of the control
- **Creating an anonymous inner class:**

  1. An anonymous class is defined with an OnClickListener interface, and an onClick(View v) method is implemented in it.

  2. The anonymous inner class is passed to the listener through the setOnClickListener() method.
- **Implementing the OnClickListener interface:**

  1. The Activity class implements the OnClickListener interface, which requires us to implement the onClick() method in this class.

  2. The onClick() method is declared inside the class, and the listener is set by passing a reference to the class by the following statement: btn.setOnClickListener();

- **Declaring the event handler in the XML definition of the control:**

  1. In the XML definition of a Button in the layout file, you can add an android:onClick attribute. This attribute is used to represent the method you want to execute when a click event occurs via the Button control.

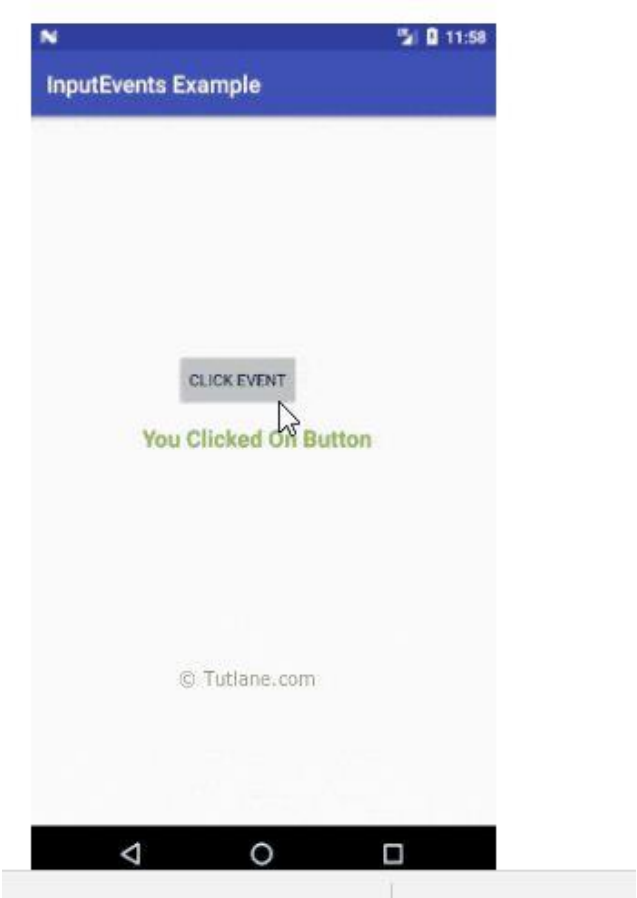
- **activity_main.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >
   <Button
      android:id="@+id/btnClick"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Click Event"
      android:layout_marginTop="200dp" android:layout_marginLeft="130dp"/>
<TextView
      android:id="@+id/txtResult"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="100dp"
      android:textColor="#86AD33"
      android:textSize="20dp"
      android:textStyle="bold"
      android:layout_marginTop="12dp"/>
</LinearLayout>
```

- **MainActivity.java**

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

```
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity

{
   Button btn;
   TextView tView;
   @Override
   protected void onCreate(Bundle savedInstanceState)

 {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      btn = (Button)findViewById(R.id.btnClick);
      tView = (TextView)findViewById(R.id.txtResult);
      btn.setOnClickListener(new View.OnClickListener()

                    {
                     @Override
                public void onClick(View v)

                     {
                          tView.setText("You Clicked On Button");
                      }
                });
          }
}
```



Displaying Messages through toast:

- A Toast is a transient message that automatically disappears after a while without user interaction.
- A Toast is created by calling the static method, makeText(), of theToast class. The syntax of the makeText() method is shown here:

  Toast.makeText(context,text, duration)toast.show();
- The duration is expressed in the form of constants, such as Toast.LENGTH_SHORT or Toast.LENGTH_LONG,

Example :

Xml file:

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
    android:orientation="vertical" >

   <Button
    android:id="@+id/mainbutton“
    android:layout_width="wrap_content"
   android:layout_height="wrap_content"
    android:text="@string/button_label" />
</LinearLayout>
```

Java file:

```
package com.javacodegeeks.android.androidtoastexample;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;
public class MainActivity extends Activity {
   private Button button;
  public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
 setContentView(R.layout.main);
button = (Button) findViewById(R.id.mainbutton);
       button.setOnClickListener(new OnClickListener() {
          @Override
          public void onClick(View arg0) {
            Toast.makeText(getApplicationContext(),"This is an Android Toast Message",
Toast.LENGTH_LONG).show();
        }
      });
    }
}
```

Creating and staring an activity:

Activity is to make the project interactive. Here this can be happen with the help of Intent. Intent is a data structure consists of action that application needs to perform and also used to transfer data between activities.

For example: Intent facilitate you to redirect your activity to another activity on occurrence of any event. By calling, startActivity() you can perform this task.

```
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

In the above example, foreground activity is getting redirected to another activity i.e. SecondActivity.java. getApplicationContext() returns the context for your foreground activity.

Intents are generally classified as 2 types:-

Explicit Intent:

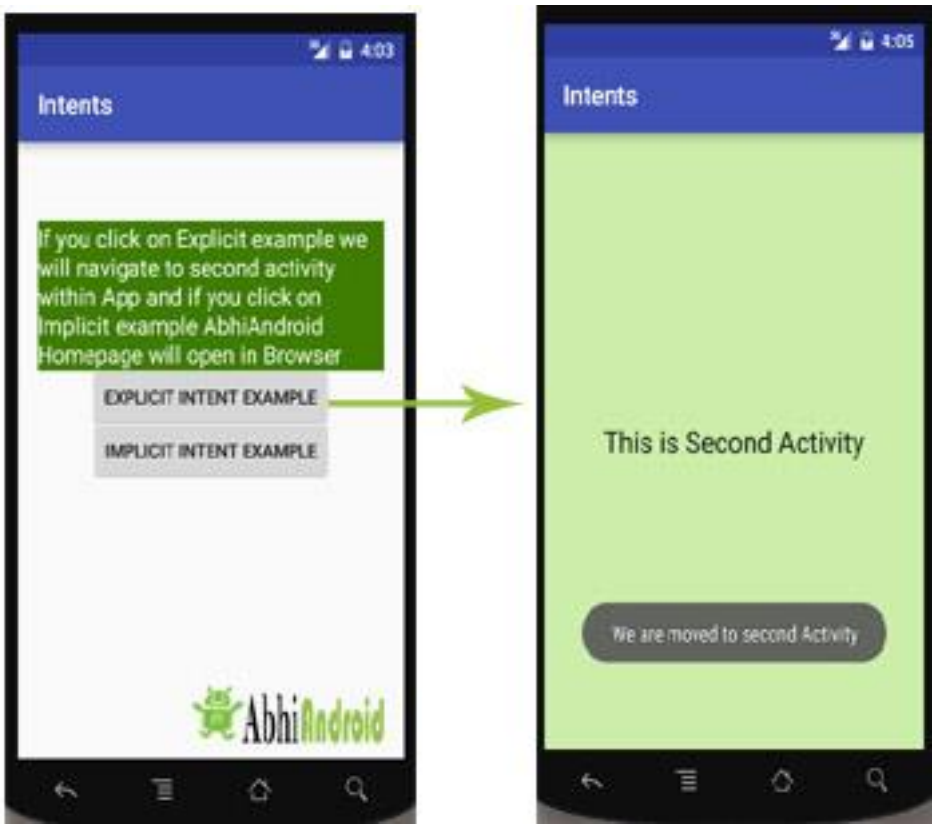
- Explicit Intents are used to connect the application internally.
- In Explicit we use the name of component which will be affected by Intent. For Example: If we know class name then we can navigate the app from One Activity to another

activity using Intent. In the similar way we can start a service to download a file in background process.

Explicit Intent work internally within an application to perform navigation and data transfer. The below given code snippet will help you understand the concept of Explicit Intents

```
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

Here SecondActivity is the JAVA class name where the activity will now be navigated. Example with code in the end of this post will make it more clear.



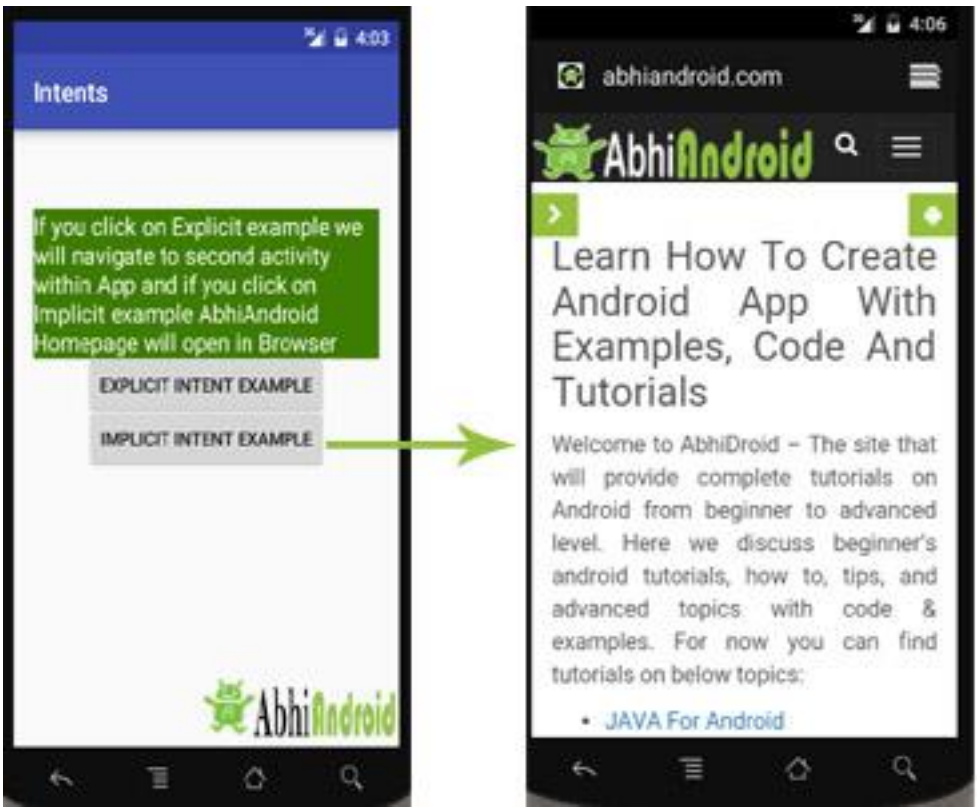
Implicit Intent:

- In Implicit Intents we do need to specify the name of the component. We just specify the Action which has to be performed and further this action is handled by the component of another application.
- The basic example of implicit Intent is to open any web page

Let’s take an example to understand Implicit Intents more clearly. We have to open a website using intent in your application. See the code snippet given below

```
Intent intentObj = new Intent(Intent.ACTION_VIEW);
intentObj.setData(Uri.parse("https://www.abhiandroid.com"));
startActivity(intentObj);
```

Unlike Explicit Intent you do not use any class name to pass through Intent(). In this example we has just specified an action. Now when we will run this code then Android will automatically start your web browser and it will open AbhiAndroid home page.



Intent Example in Android:

Let’s implement Intent for a very basic use. In the below example we will Navigate from one Activity to another and open a web homepage of AbhiAndroid using Intent. **The example will show you both implicit and explicit Intent together.**

Create a project in Android Studio and named it “Intents”. Make an activity, which would consists Java file; MainActivity.java and an xml file for User interface which would be activity_main.xml

Step 1: Let’s design the UI of activity_main.xml:

- First design the text view displaying basic details of the App
- Second design the two button of Explicit Intent Example and Implicit Intent Example

Below is the complete code of activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="If you click on Explicit example we will navigate to second activity within App and if you click on Implicit example AbhiAndroid Homepage will open in Browser"
        android:id="@+id/textView2"
        android:clickable="false"
        android:layout_alignParentTop="true"
```

```
        android:layout_alignParentStart="true"
        android:layout_marginTop="42dp"
        android:background="#3e7d02"
        android:textColor="#ffffff" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Explicit Intent Example"
        android:id="@+id/explicit_Intent"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="147dp" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Implicit Intent Example"
        android:id="@+id/implicit_Intent"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Step 2: Design the UI of second activity activity_second.xml

Now lets design UI of another activity where user will navigate after he click on Explicit Example button. Go to layout folder, create a new activity and name it activity_second.xml.

- In this activity we will simply use TextView to tell user he is now on second activity.

Below is the complete code of activity_second.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:background="#CCEEAA"
    tools:context="com.example.android.intents.SecondActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="This is Second Activity"
        android:id="@+id/textView"
        android:layout_centerVertical="true"
```

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

</RelativeLayout>
```

Step 3: Implement onClick event for Implicit And Explicit Button inside MainActivity.java

Now we will use setOnClickListener() method to implement OnClick event on both the button. Implicit button will open AbhiAndroid.com homepage in browser and Explicit button will move to SecondActivity.java.

Below is the complete code of MainActivity.java

```
package com.example.android.intents;


import android.content.Intent;

import android.net.Uri;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;


public class MainActivity extends AppCompatActivity {



   Button explicit_btn, implicit_btn;

   @Override

   protected void onCreate(Bundle savedInstanceState) {

      super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.activity_main);
explicit_btn = (Button)findViewById(R.id.explicit_Intent);
implicit_btn = (Button) findViewById(R.id.implicit_Intent);

//implement Onclick event for Explicit Intent

explicit_btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        Intent intent = new  Intent(getBaseContext(), SecondActivity.class);
        startActivity(intent);
    }
});

//implement onClick event for Implicit Intent

implicit_btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.setData(Uri.parse("https://www.abhiandroid.com"));
```

```
            startActivity(intent);
        }
    });
  }
}
```

Step 4: Create A New JAVA class name SecondActivity

Now we need to create another SecondActivity.java which will simply open the layout of activity_second.xml . Also we will use Toast to display message that he is on second activity.

Below is the complete code of SecondActivity.java:

```
package com.example.android.intents;


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


public class SecondActivity extends AppCompatActivity {


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

```
        Toast.makeText(getApplicationContext(), "We are moved to second Activity",Toast.LENGTH_LONG).show();
    }
}
```

Step 5: Manifest file:

Make sure Manifest file has both the MainActivity and SecondActivity listed it. Also here MainActivity is our main activity which will be launched first. So make sure intent-filter is correctly added just below MainActivity.

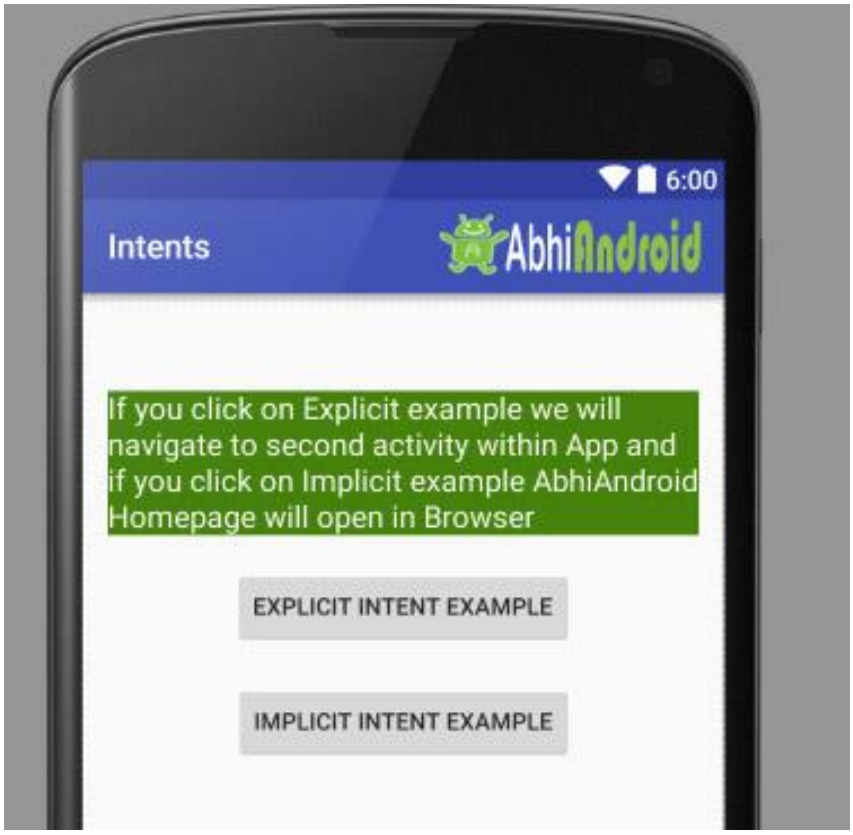
Below is the code of Manifest file:

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
```

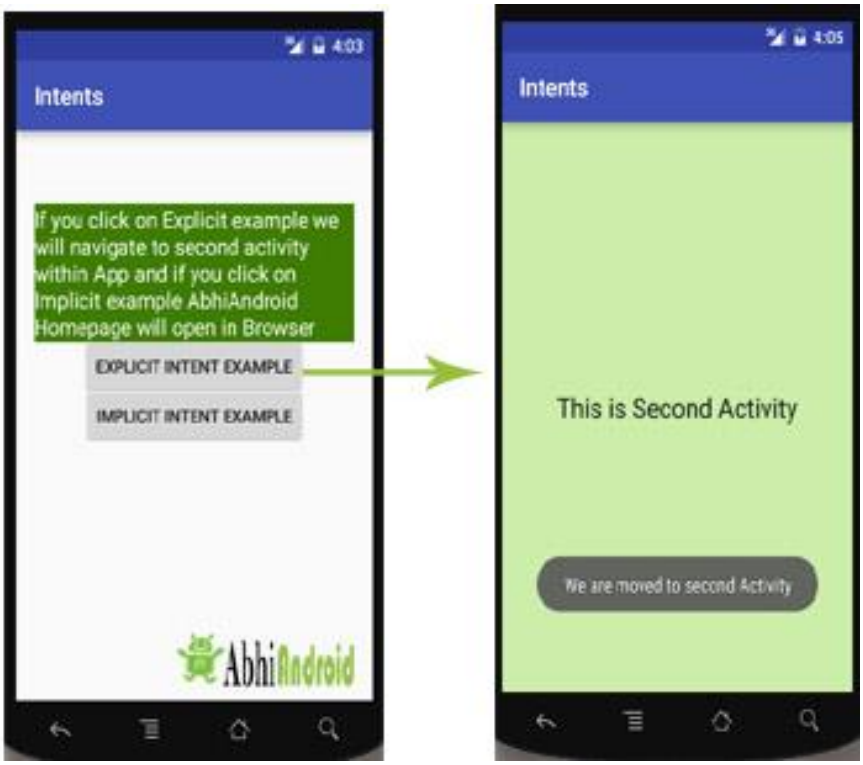
```
            </intent-filter>
        </activity>
        <activity android:name=".SecondActivity" >


        </activity>
    </application>


</manifest>
```

Output:

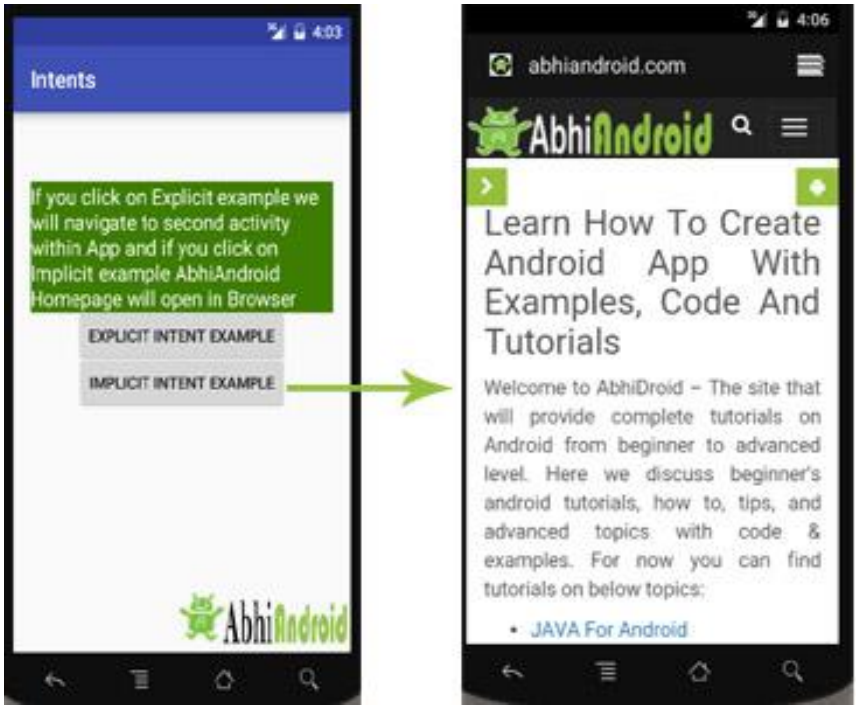
Now run the above program in your Emulator. The App will look like this:



First Click on Explicit Intent Example. The SecondActivity will be open within the App:



Now go back in Emulator and click on Implicit Intent Example. The AbhiAndroid.com homepage will open in Browser (make sure you have internet):



Using the EditText Control:

- The EditText control is a subclass of TextView and is used for getting input from the user.
- You can use several attributes to configure the EditText control to suit your requirements.
- You can implement the scrolling feature to scroll the text when the user types beyond the width of the control or the auto text feature to correct common spelling mistakes while the user types.
- The default behavior of the EditText control is to display text as a single line and run over to the next line when the user types beyond the width of the control.
- By applying attributes, you can constrain the EditText control to remain on a single line only and let the text scroll to the left.
- You can also hide the characters typed by the user for security purposes, that is, convert the characters into dots, which you usually see while typing passwords.

Attributes Used to Configure the EditText Control:

- **android:layout_width**—Used to set the width of the EditText control.
- **android:layout_height**—Used to set the height of the EditText control.
- **android:singleLine-** To have Edit text control respected to only single line.
- **android:hint-** Helpful text that used to display message on EditText by default
- **android:lines—**Sets the height of the EditText control to accommodate the specified number of lines
- **android:textSize**—Sets the size of the text typed in the EditText control.
- **android:autoText**—When set to true enables the EditText control to correct common spelling mistakes.
- **android:capitalize**—Automatically converts typed text into capital letters.
- **android:password**—When set to true, the attribute converts the typed characters into dots to hide entered text.
- **android:minWidth**—Used to specify the minimum width of the control.
- **android:maxWidth**—Used to specify the maximum width of the control.
- **android:minHeight**—Used to specify the minimum height of the control.
- **android:maxHeight**—Used to specify the maximum height of the control.
- **android:scrollHorizontally**—When set to true, makes the text scroll horizontally if typed beyond the specified width of the control.

- **android:inputType**- Specifies type of data i.e, entered in EditText and configures onscreen Keyboard. It associates with values as follows:

  textpassword, textautocorrect, number, phons,text

Example:

activity_main.xml

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

<LinearLayout 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:orientation="vertical"

android:layout_width="match_parent"

android:layout_height="match_parent"

tools:context=".MainActivity">

<EditText

android:layout_width="match_parent"

android:layout_height="wrap_content"

android:id="@+id/et" />

<Button

android:layout_width="match_parent"

android:layout_height="wrap_content"

android:text="Display" android:onClick="display" />

<TextView

android:layout_width="match_parent"

android:layout_height="wrap_content"
```

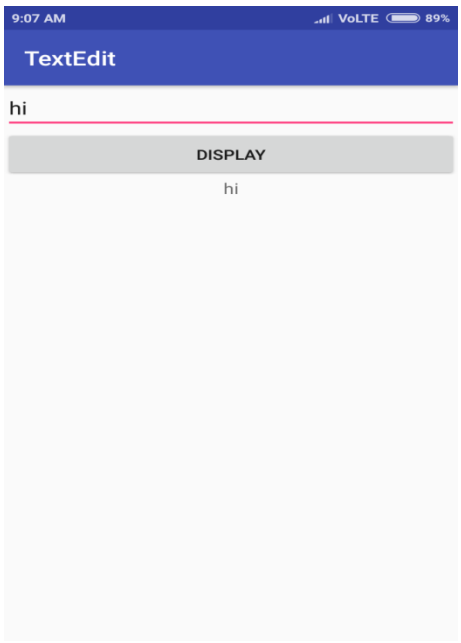
```
    android:gravity="center"
    android:text=""
    android:id="@+id/tv"
    android:textSize="15sp" />
</LinearLayout>
```

MainActivity.java.

```
package in.co.jntuaallupdates.www.textedit;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    Button btn;
    EditText editText;
    TextView textView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        editText=findViewById(R.id.et);
```

```
        textView=findViewById(R.id.tv);
    }
    public void display(View view) {
        textView.setText(editText.getText().toString());
    }
}
```

Output:



Choosing Options with CheckBox

- Checkbox is used for selection control.
- Selection of multiple from group of values.
- The selection of value/option is treated as “checked state”, unselection is treated as “unchecked”.

Methods:

ischecked(): to determine status of checkbox

setchecked(): changes the state of checkbox

toggle()—Toggles the state of the check box from checked to unchecked.

- To add an event listener, you can implement the OnCheckedChangeListener interface that invokes the callback method onCheckedChanged() when the state of the check box changes.
- you can also use the traditional implementation, the OnClickListener interface, that invokes the onClick() method when any of the CheckBox controls are clicked.

Example:

activity_main.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Select Items you want"/>
<CheckBox
android:id="@+id/checkbox_pizza"
android:layout_height="wrap_content"
android:text="Pizza $15"
android:layout_width="match_parent" />
<CheckBox
android:id="@+id/checkbox_hotdog"
android:layout_height="wrap_content"
android:text="Hot Dog $5"
android:layout_width="match_parent" />
<CheckBox
android:id="@+id/checkbox_burger"
android:layout_height="wrap_content"
android:text="Burger $10"
android:layout_width="match_parent" />
<Button
```

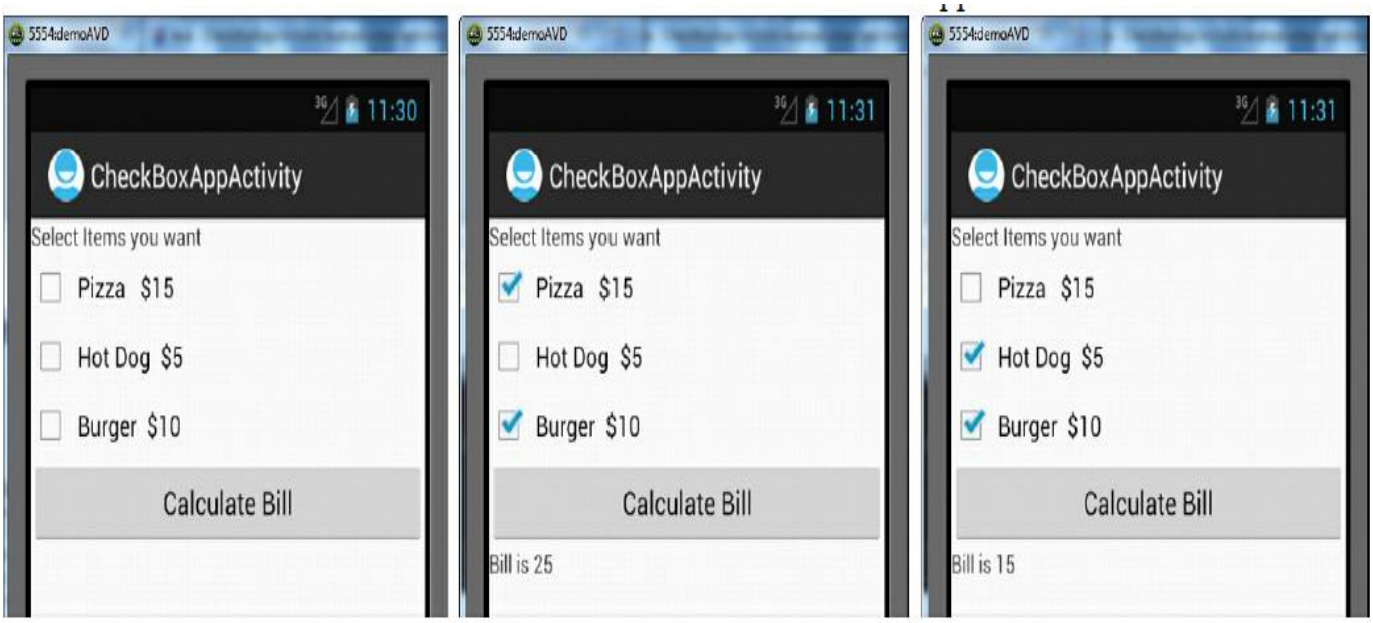
```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/bill_btn"
android:text="Calculate Bill"/>
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/amount"/>
</LinearLayout
```

MainActivity.java.

```
package com.androidunleashed.checkboxapp;
import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import android.widget.CheckBox;
import android.view.View;
import android.view.View.OnClickListener;
public class CheckBoxAppActivity extends Activity implements
OnClickListener {
CheckBox c1,c2,c3;
TextView resp;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_check_box_app);
Button b = (Button)this.findViewById(R.id.bill_btn);
resp = (TextView)this.findViewById(R.id.amount);
c1 = (CheckBox)findViewById(R.id.checkbox_pizza);
c2 = (CheckBox)findViewById(R.id.checkbox_hotdog);
c3 = (CheckBox)findViewById(R.id.checkbox_burger);
b.setOnClickListener(this);
```

```
}
public void onClick(View v) {
int amt=0;
if (c1.isChecked()) {
amt=amt+15;
}
if (c2.isChecked()) {
amt=amt+5;
}
if (c3.isChecked()) {
amt=amt+10;
}
resp.setText("Bill is " +Integer.toString(amt));
}
}
```

Output:



Choosing Mutually Exclusive Items Using RadioButtons

RadioButton control is also used for selecting the value from group of values as like checkbox, but the difference is only one option can be selecting from group of values in Radiobutton, Hence it is called as mutually exclusive.

Whenever radiobutton is selected if anyother button in that group has been already selected will got deselected i.e, unchecked. Hence these are treated as mutually exclusive items. Here to make the radiobutton control as mutually exclusive we have to declare the group as Radiogroup for that Radiogroup element is used.

Ischecked( ):- checks whether Radiobutton is selected/ not

Toggle( ):- changes the Radiobutton whose ID is supplied.

Check( ):- checks the Radiobutton whose ID is supplied.

getCheckedRadioButtonId( ):- gets the Id of currently selected RadiobuttonControl

Example:

activity_main.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Select the type of hotel"/>
<RadioGroup
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="vertical">
<RadioButton android:id="@+id/radio_fivestar"
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
android:text="Five Star " />
<RadioButton android:id="@+id/radio_threestar"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Three Star" />
</RadioGroup>
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/hoteltype"/>
</LinearLayout>
```

MainActivity.java.

```
package com.androidunleashed.radiobuttonapp;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.RadioButton;
import android.view.View;
import android.view.View.OnClickListener;
public class RadioButtonAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_radio_button_app);
RadioButton radioFivestar = (RadioButton)
findViewById(R.id.radio_fivestar);
RadioButton radioThreestar = (RadioButton)
findViewById(R.id.radio_threestar);
radioFivestar.setOnClickListener(radioListener);
radioThreestar.setOnClickListener(radioListener);
}
```

```
private OnClickListener radioListener = new
OnClickListener() {
public void onClick(View v) {
TextView selectedHotel = (TextView)
findViewById(R.id.hoteltype);
RadioButton rb = (RadioButton) v;
selectedHotel.setText("The hotel type selected is: "
+rb.getText());
}

};

}
```

Output: