Getting Started

Today's Overview

1. The Tools
2. Android Project Structure
3. Android Application Lifecycle
4. Android UI

The Tools

- All tools needed for Android app development are free and can be downloaded from the Web
- There are 2 basic tools needed:
  1. JAVA JDK – the Android SDK makes use of the Java SE Development Kit (JDK) for Java code compilation

The Tools

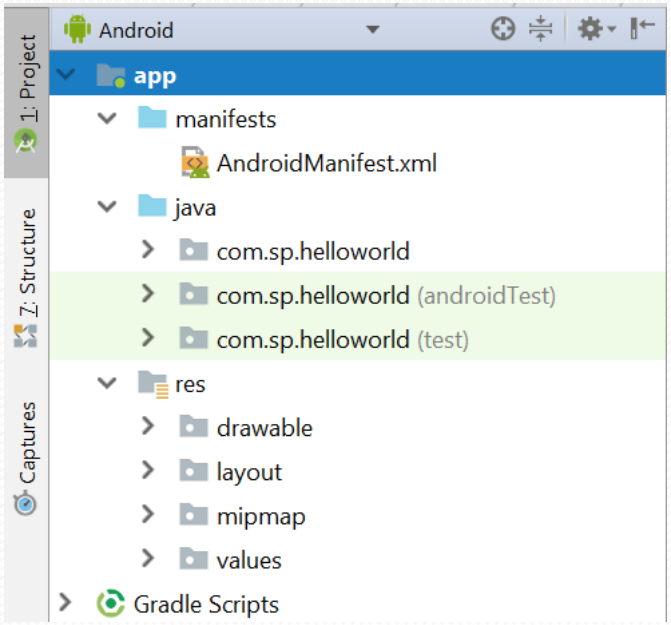
- All tools needed for Android app development are free and can be downloaded from the Web
- There are 2 basic tools needed:
  2. Android Studio – the official IDE (Integrated Development Environment) for Android application development. It is integrated with a debugger, libraries, emulator, documentation, sample code, and tutorials.

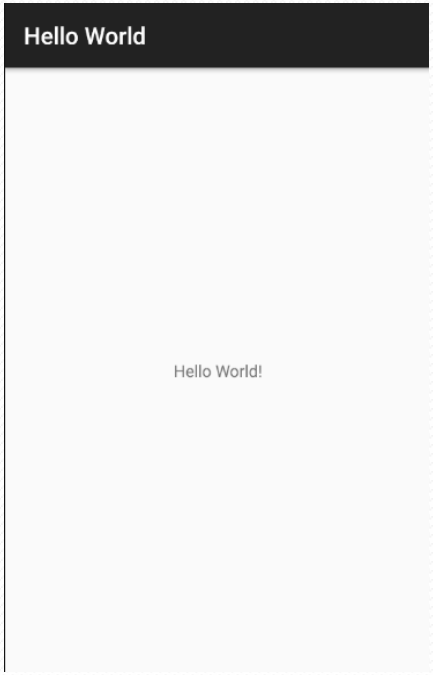
Environment Set-up

- In order for the development environment to work, you will need to follow the set-up sequence below:
  1. Install the Java JDK
  2. Install the Android Studio
  3. Create required Android Virtual Device (AVD) with necessary System Image downloaded from developer server

Environment Testing

- To check on the functionality of the Android development environment set-up, Android Studio allows user to create a standard Android “HelloWorld” project for testing
- If successful, a phone emulator (AVD) will be launched as shown





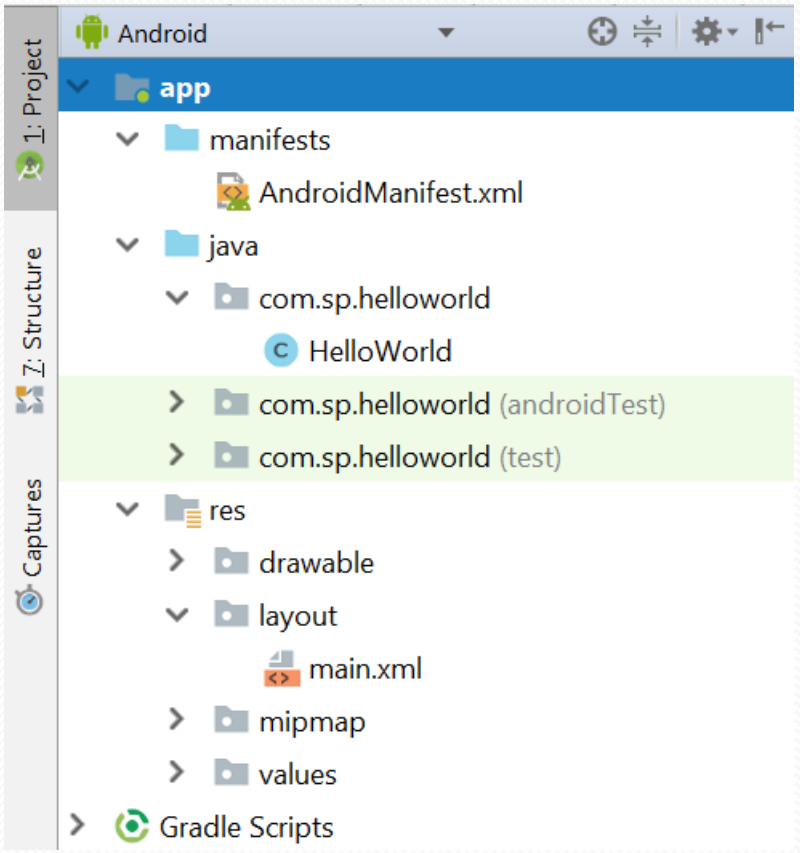
Understanding Hello World

Android Studio Project Structure

Android Studio Project Structure

The main folders contains in standard Android project created by Android Studio

- Folders
  - manifests
  - java
  - res



Project Folder Structure

FOLDER	EXPLANATION
manifests	The '*AndroidManifest.xml*' file contains essential information about your app that the Android system needs. This includes the activities and services your app uses, the permissions it requires, any intends it responds to, and basic info like the app name and app icon
java	Contains app Java's program code
res	Contains all resources required in the app • drawable – contains images used in the app • layout – contains all XML files which define the view presented on Android device • menu – contains all XML files which define MENU items • mipmap – contains app launcher icon in different resolutions • values – contains definition files for device dimension, style for themes and variables use in XML files

Hello World

- *Line 6*

  HelloWorld is declared as a sub-class of *Activity* class. It inherits all properties & methods from Activity class

```
1   package com.sp.helloworld;
2
3   import android.app.Activity;
4   import android.os.Bundle;
5
6   public class HelloWorld extends Activity {
7
8       @Override
9       protected void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11          setContentView(R.layout.main);
12      }
13  }
```

Hello World

- *Line 9*

When Hello World App is first launched, *HelloWorld* Activity is created. At the beginning of application lifecycle, **onCreate(Bundle)** callback method will be run and codes within this method will be executed first

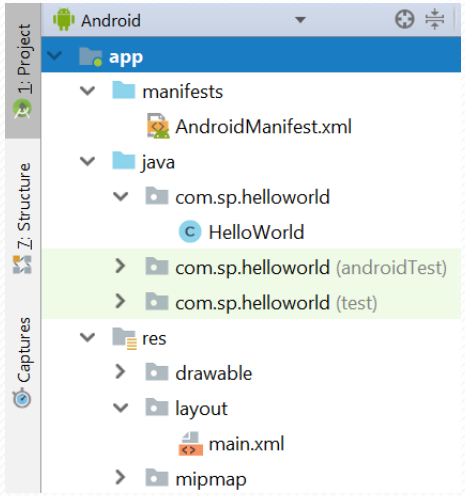
```
1   package com.sp.helloworld;
2
3   import android.app.Activity;
4   import android.os.Bundle;
5
6   public class HelloWorld extends Activity {
7
8       @Override
9       protected void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11          setContentView(R.layout.main);
12      }
13  }
```

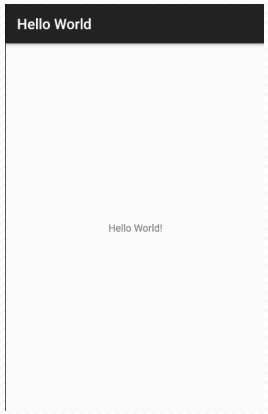
Hello World

- *Line 11*

The **setContentView(R.layout.main)** will set the current display with the layout defined by *main.xml* in the res/layout folder

```
1   package com.sp.helloworld;
2
3   import android.app.Activity;
4   import android.os.Bundle;
5
6   public class HelloWorld extends Activity {
7
8       @Override
9       protected void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11          setContentView(R.layout.main);
12      }
13  }
```

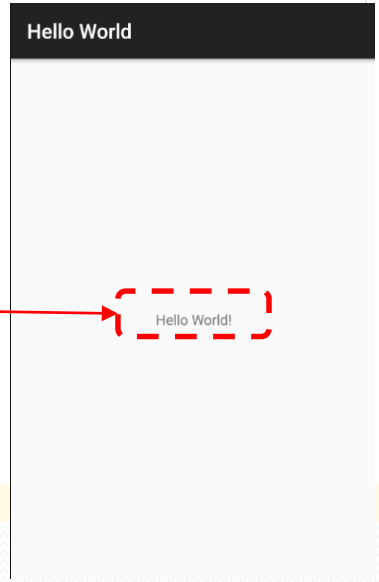




Hello World

- The *main.xml* layout defines the UI in *XML*
- *XML* stands for Extensible Markup Language. It was designed to carry data

```
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=".HelloWorld">
8
9      <TextView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Hello World!"
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
18 </android.support.constraint.ConstraintLayout>
```



Listing – main.xml

Hello World

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

Root Element

```
<?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=".HelloWorld">

    <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>
```

Listing – main.xml

Hello World

- Each element will begin with a "start" tag and will end with a "end" tag
- In between the "start" and "end" tag, you can have text content or more elements ("child" elements)
- Example below shows that there are two elements :

  "*ConstraintLayout*" – parent or root element

  "*TextView*" – child

```
<ConstraintLayout>
    <TextView />
</ConstraintLayout>
```

```
<?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=".HelloWorld">

    <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>
```

Hello World

- In each element, attributes can be added to define the characteristic of the element
- Example

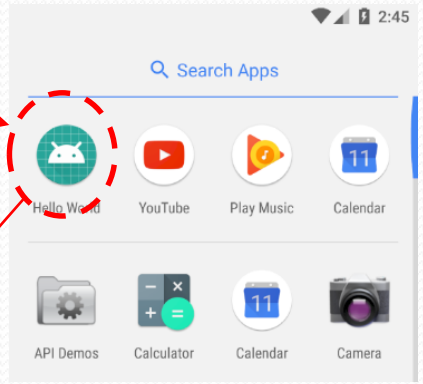
```
1  <?xml version="1.0" encoding="utf-8"?>
2  <android.support.constraint.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".HelloWorld">
9
10     <TextView
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:text="Hello World!"
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>
```

Listing – main.xml

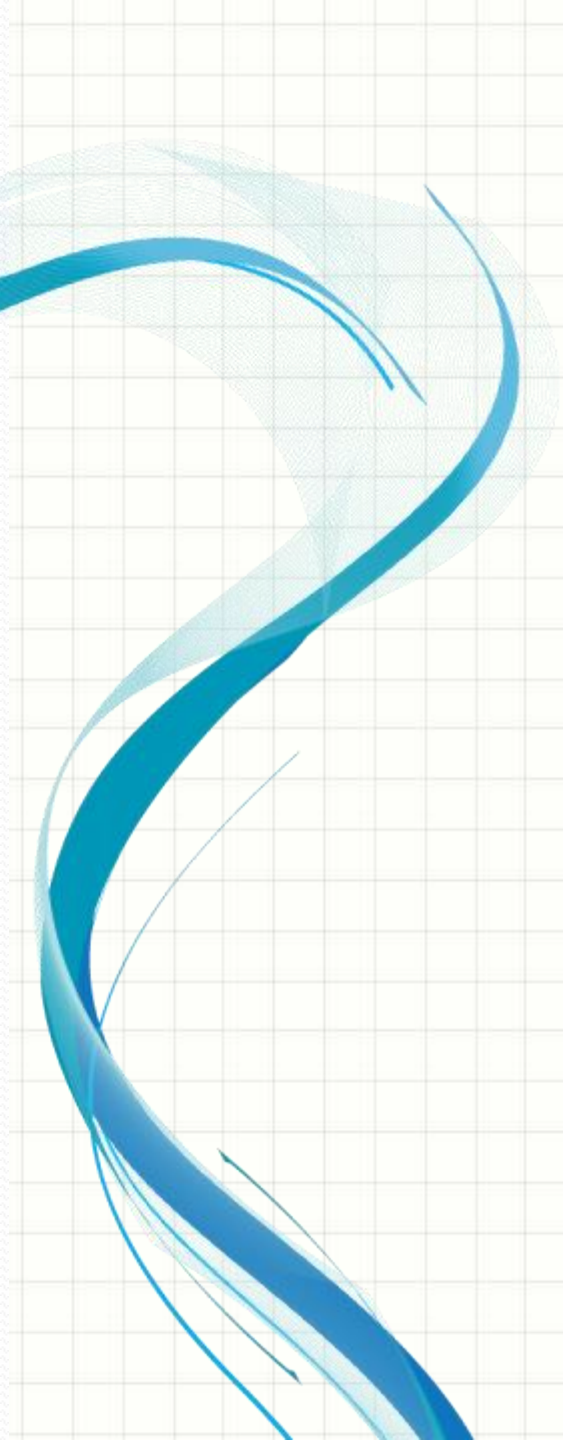
Hello World

- The *AndroidManifest.xml* file defines all Activities in the app. It also defines the main launch Activity. A runtime exception will occur if trying to start any Activity not defined in the file

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      package="com.sp.helloworld">
5
6      <application
7          android:allowBackup="true"
8          android:icon="@mipmap/ic_launcher"
9          android:label="Hello World"
10         android:roundIcon="@mipmap/ic_launcher_round"
11         android:supportsRtl="true"
12         android:theme="@style/AppTheme">
13         <activity android:name=".HelloWorld">
14             <intent-filter>
15                 <action android:name="android.intent.action.MAIN" />
16
17                 <category android:name="android.intent.category.LAUNCHER" />
18             </intent-filter>
19         </activity>
20     </application>
21
22 </manifest>
```



Listing – AndroidManifest.xml



Application Distribution File

Android Application Package (APK)

- Android application package file (APK) is the file format used to distribute and install application software onto Android device
- APK files are ZIP file formatted packages based on the JAR file format, with .apk file extensions
- It can be found in sub-folder "***app/build/outputs/apk***" of the project folder create by Android Studio

Android Application Package (APK)

- An APK file is an archive that usually contains the following folders and files:
  1. META-INF directory
     - MANIFEST.MF: the Manifest file
     - CERT.RSA: The certificate of the application.
     - CERT.SF: The list of resources and SHA-1 digest
  2. res directory containing resources files e.g. images, icon

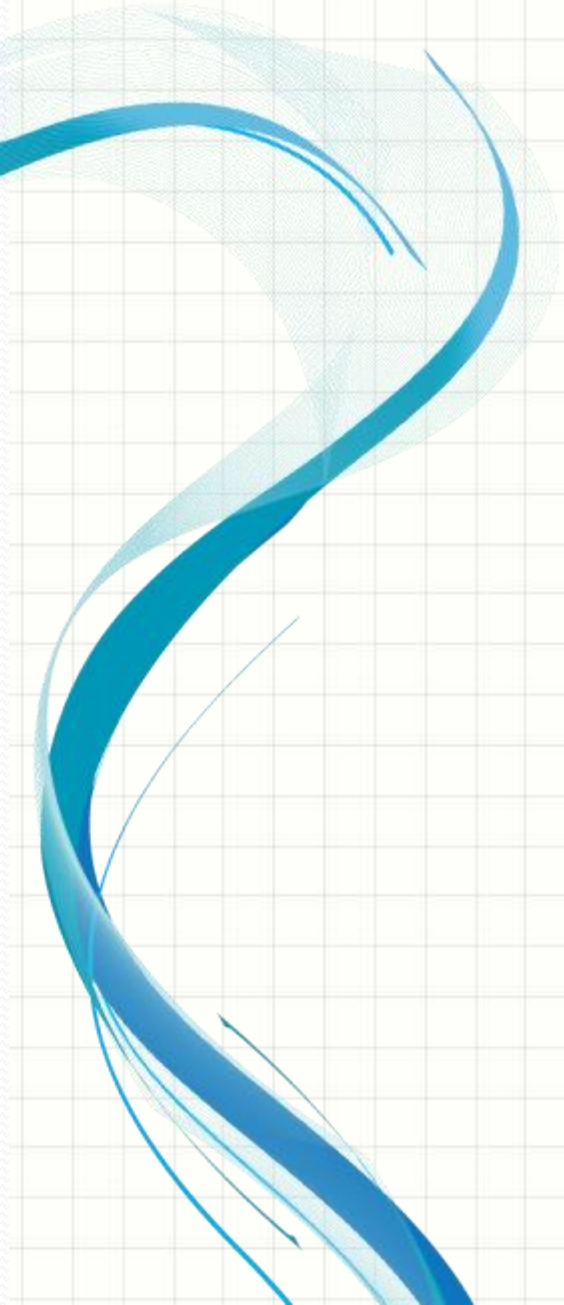
Android Application Package (APK)

3. AndroidManifest.xml: An additional Android manifest file, describing the name, version, access rights, referenced library files for the application
4. classes.dex: The classes compiled in the dex format understandable by the Dalvik virtual machine
5. resources.arsc : a file containing pre-compiled resources, such as binary XML for example

Android Application Package (APK)

- Each APK installed on an Android device is given its own Linux user ID, and this ID remains unchanged for as long as the APK resides on the device
- Security enforcement occurs at the process level, so the code contained in any two APKs cannot normally run in the same process, because each APK's code needs to run as a different Linux user

Application Launched



Activity Lifecycle

Activity Lifecycle

- Android OS allows multiple apps to run concurrently
- There can be only one active app visible to user at a time – specifically, a single app Activity is in the foreground at any given time
- Android OS keeps track of all Activity objects running by placing them on an Activity stack

Activity Lifecycle

- Activity Stack

  The state of each Activity is determined by its position on the Activity stack, a last-in-first-out collection of all the currently running Activities

  When a new Activity starts, the current foreground screen is moved to the top of the stack

  If the user navigates back using the BACK button, or the foreground Activity is closed, the next Activity on the stack moves up and becomes active

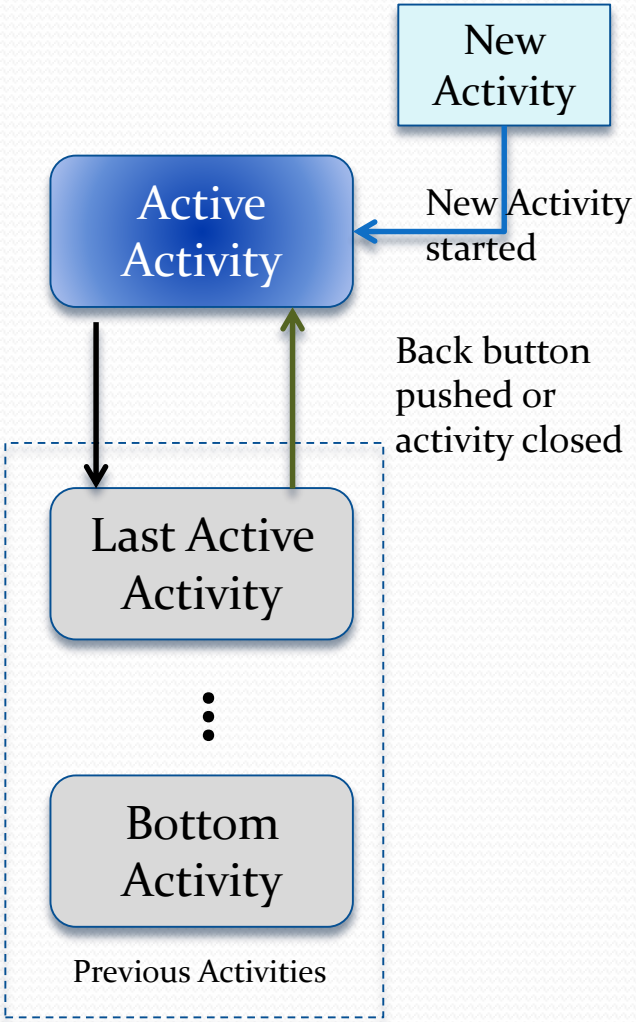
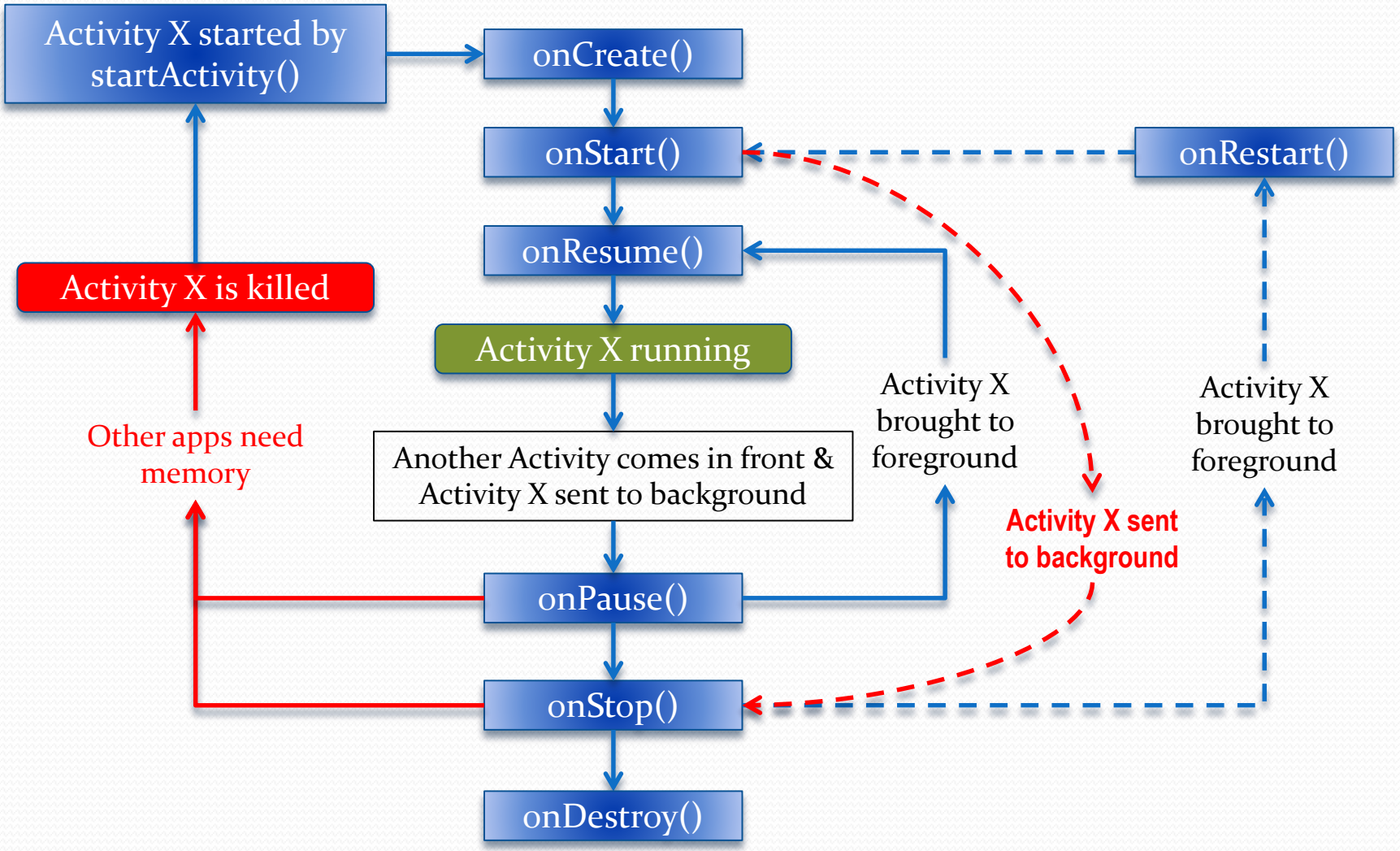


Fig – The Activity stack

Activity Lifecycle

- To ensure that Activities can react to state changes, Android provides a series of event handlers (or activity callbacks) that are fired when an Activity transitions through its full, visible, and active
- Here are the 7 callback methods of Activity class:
  - protected void onCreate(Bundle savedInstanceStates)
  - protected void onStart()
  - protected void onRestart()
  - protected void onResume()
  - protected void onPause()
  - protected void onStop()
  - protected void onDestroy()

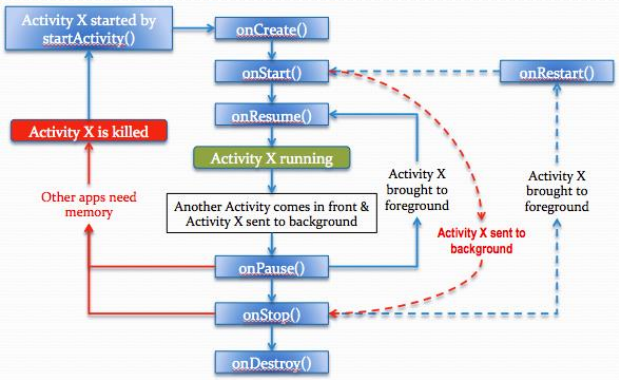
Activity Lifecycle



Activity Lifecycle

- onCreate(Bundle)

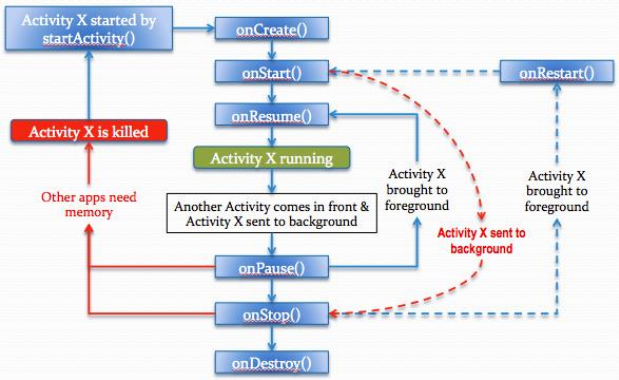
  It is called when the activity first created. It is used to create the activity's UI, background threads as needed, and perform other global initialization. onCreate() is passed an android.os.Bundle object containing the activity's previous state, if that state was captured; otherwise, a null reference is passed. Android always calls onStart() method after calling onCreate(Bundle)



Activity Lifecycle

- onStart()

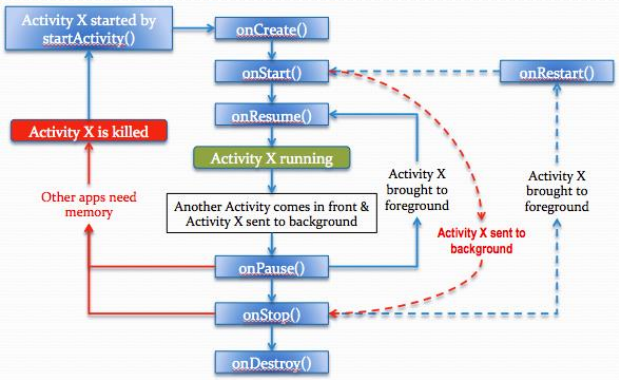
  This method is called just before the activity becomes visible to the user. Android calls the onResume() method after calling onStart() when the activity comes to foreground, and calls the onStop() method after onStart() when the activity becomes hidden



Activity Lifecycle

- onRestart()

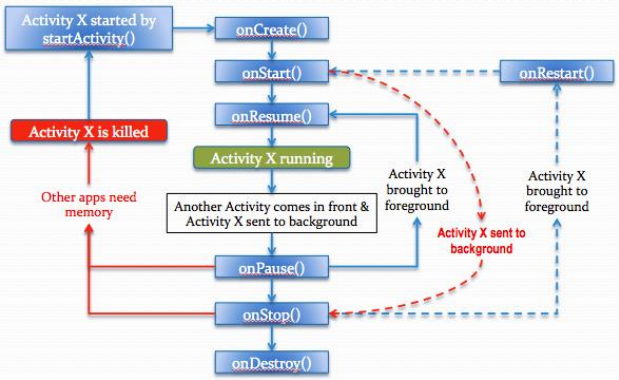
  This method is called after the activity has been stopped, just prior to it being started again. Android always calls onStart() after calling onRestart()



Activity Lifecycle

- onResume()

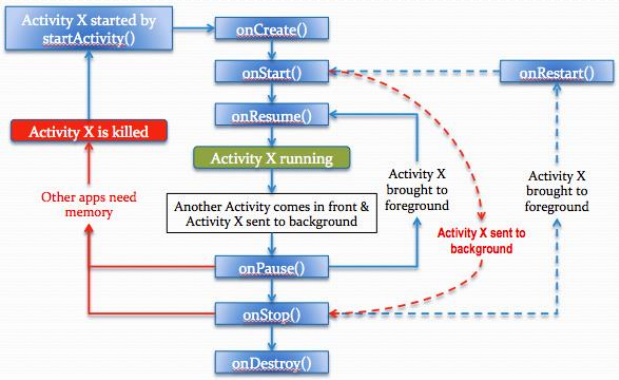
  This method is called just before the activity starts interacting with the user. Android always calls the onPause() method after calling onResume(), but only when the activity must be paused



Activity Lifecycle

- onPause()

  This method is called when Android is about to resume another activity. This method is typically used to persist unsaved changes, stop animations that might be consuming processor cycles, and so on. Android calls onResume() after calling onPause() when activity starts interacting with the user, and calls onStop() when the activity becomes invisible to the user



Activity Lifecycle

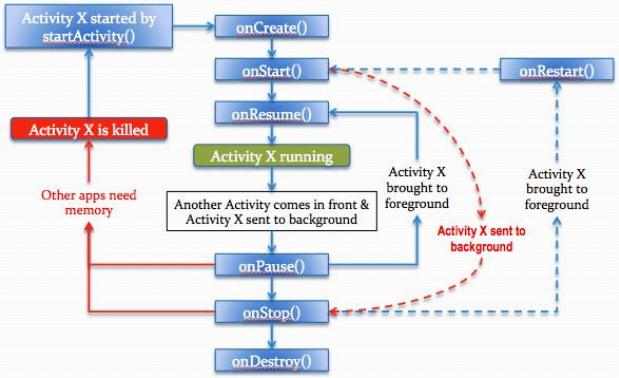
- onStop()

  This method is called when the activity is no longer visible to the user. This may happen because the activity is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering the activity. Android calls onRestart() after calling onStop(), when the activity is coming back to interact with the user, and calls the onDestroy() method when the activity is going away

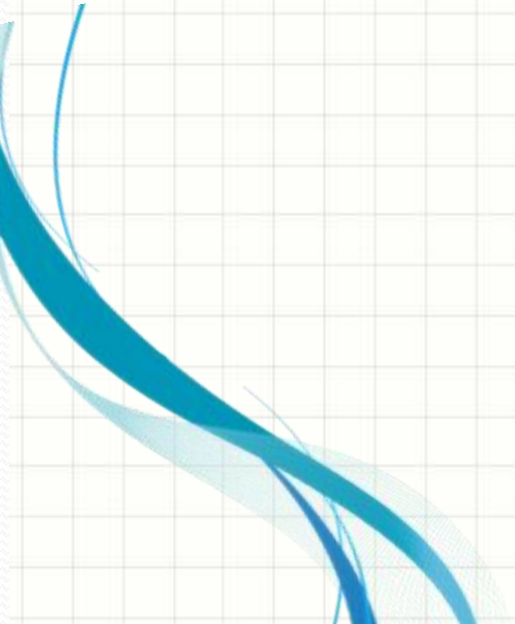
Activity Lifecycle

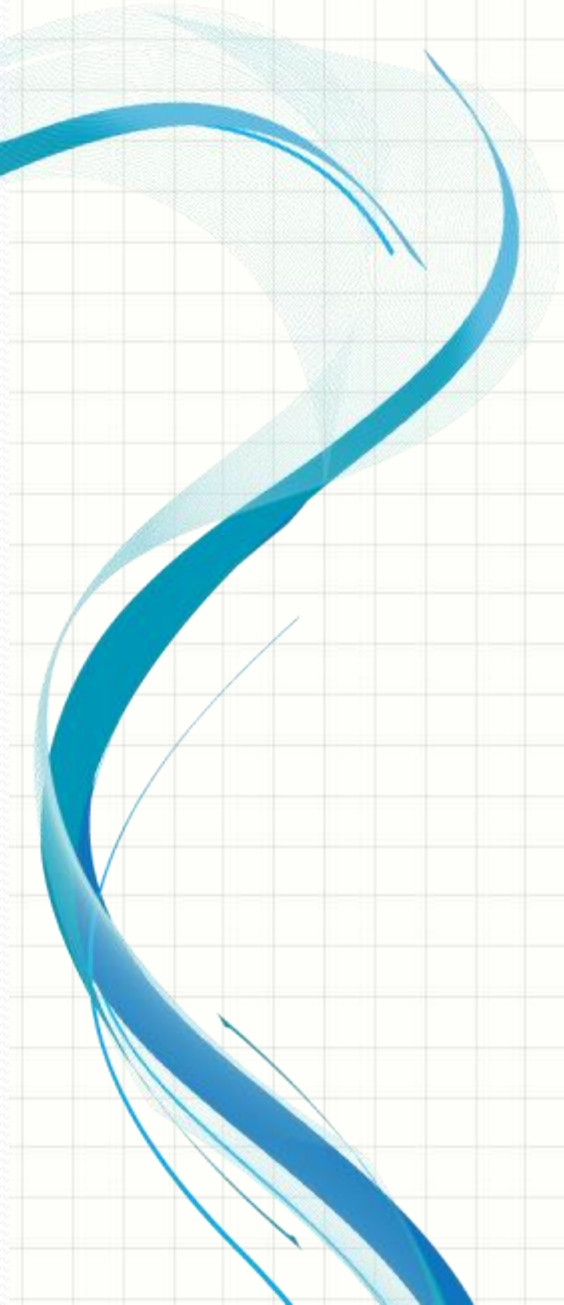
- onDestroy()

  This method is called before the activity is destroyed, unless memory is tight and Android is forced to kill the activity's process. In this scenario, onDestroy() is never called



Application UI View Design





Android Layout

Android Layout

- An Android layout is a type of resource that defines what is drawn on the screen. A layout resource is simply a template for a user interface screen, or portion of a screen
- Android user interfaces can be defined as layout resources in XML or created programmatically

Android Layout

- Using layouts to create your screens is best-practice UI design in Android
- Each layout definition is stored in a separate file, each containing a single layout, in the res/layout folder. The filename then becomes the resource identifier

Android Layout

- The standard Layouts are:
  - AbsoluteLayout
  - FrameLayout
  - LinearLayout
  - RelativeLayout
  - TableLayout

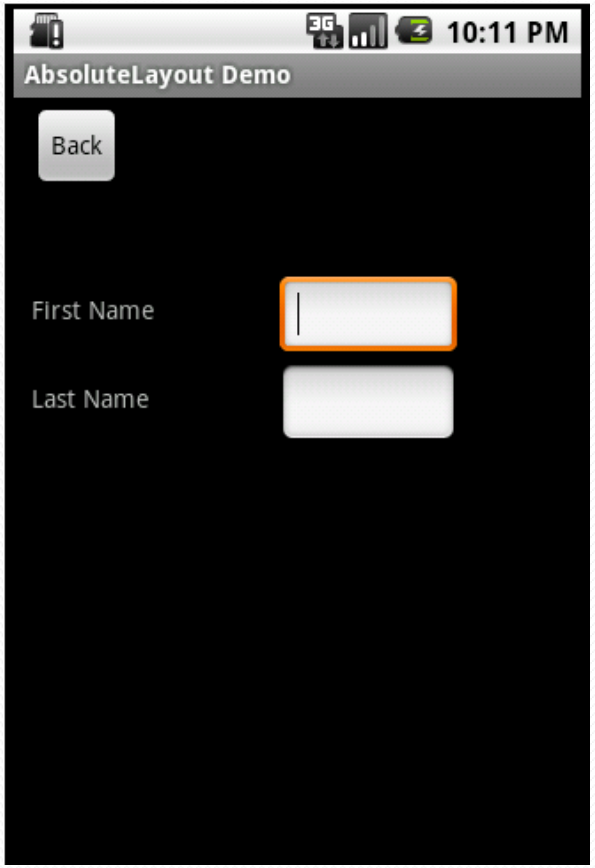
AbsoluteLayout

- AbsoluteLayout is based on the simple idea of placing each control at an absolute position. You specify the exact x and y coordinates on the screen for each control. This is not recommended for most UI development (in fact AbsoluteLayout is currently deprecated) since absolutely positioning every element on the screen makes an inflexible UI that is much more difficult to maintain. Consider what happens if a control needs to be added to the UI. You would have to change the position of every single element that is shifted by the new control.

AbsoluteLayout

- Example

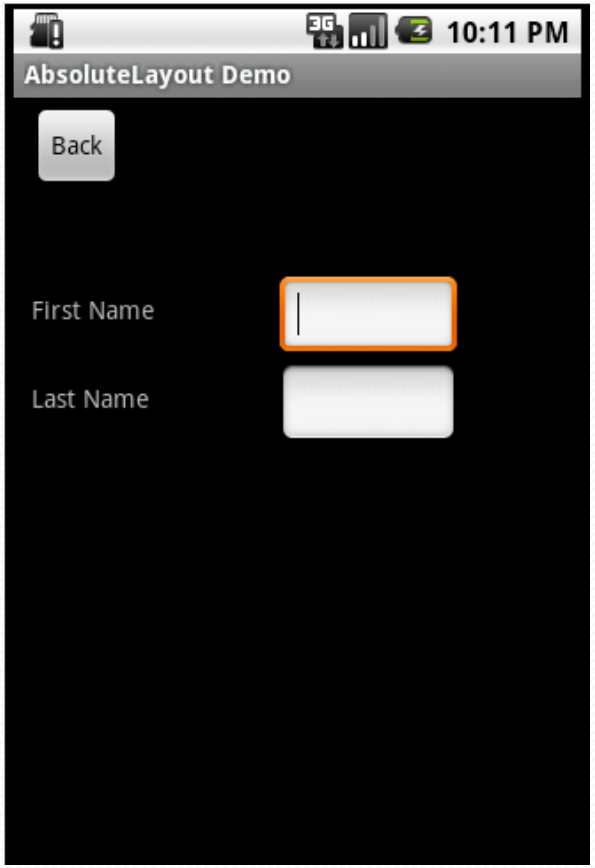
```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <Button
                android:id="@+id/backbutton"
                android:text="Back"
                android:layout_x="10px"
                android:layout_y="5px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
   <TextView
                android:layout_x="10px"
                android:layout_y="110px"
                android:text="First Name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
   <EditText
                android:layout_x="150px"
                android:layout_y="100px"
                android:width="100px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
```



AbsoluteLayout

- Example

```
<TextView
        android:layout_x="10px"
        android:layout_y="160px"
        android:text="Last Name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
<EditText
        android:layout_x="150px"
        android:layout_y="150px"
        android:width="100px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</AbsoluteLayout>
```



Android Layout

- The standard Layouts are:
  - AbsoluteLayout
  - FrameLayout
  - LinearLayout
  - RelativeLayout
  - TableLayout

FrameLayout

- FrameLayout is designed to display a single item at a time. You can have multiple elements within a FrameLayout but each element will be positioned based on the top left of the screen. Elements that overlap will be displayed overlapping

FrameLayout

- Example

```
<FrameLayout
    android:layout_width="fill_parent”
    android:layout_height="fill_parent”
    xmlns:android=http://schemas.android.com/apk/res/android>
    <ImageView
        android:src="@drawable/icon"
        android:scaleType="fitCenter"
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"/>
    <TextView
        android:text="Learn-Android.com"
        android:textSize="24sp”
        android:textColor="#oooooo”
        android:layout_height="fill_parent”
        android:layout_width="fill_parent”
        android:gravity="center"/>
</FrameLayout>
```

FrameLayout

- You can see both the *ImageView* and *TextView* fill the parent in both horizontal and vertical layout. Gravity specifies where the text appears within its container and set to center. By default, the text would have appeared at the top left of the screen.
- FrameLayout can become more useful when elements are hidden and displayed programmatically. You can use the attribute android:visibility in the XML to hide specific elements. You can call setVisibility from the code to accomplish the same thing. The three available visibility values are visible, invisible (does not display, but still takes up space in the layout), and gone (does not display, and does not take space in the layout)

FrameLayout

- You could, for example, have a game in a *FrameView* where text displayed to the user is visible in the middle of the screen at appropriate times (e.g. “Game Over”)

Android Layout

- The standard Layouts are:
  - AbsoluteLayout
  - FrameLayout
  - LinearLayout
  - RelativeLayout
  - TableLayout

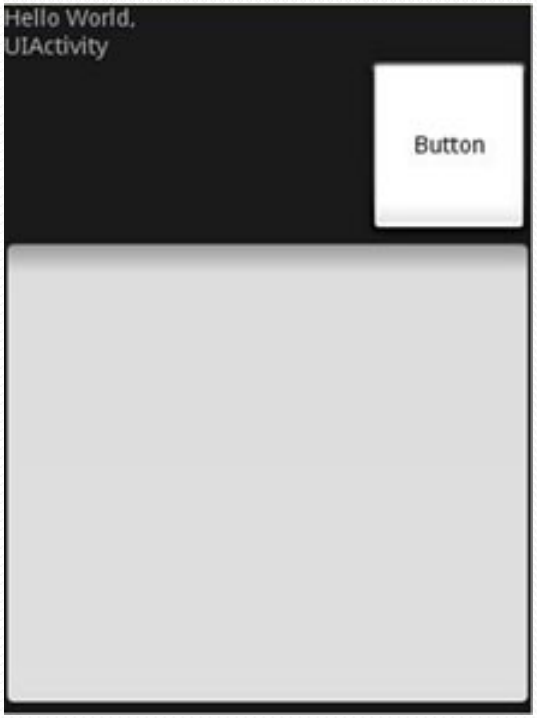
LinearLayout

- LinearLayout organizes elements along a single line. You specify whether that line is vertical or horizontal using android:orientation
- Designed to display child View controls in a single row or column. This is a very handy layout method for creating forms

LinearLayout

- Example

```
<LinearLayout
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical" >
   <TextView
      android:layout_width="105px"
      android:layout_height="wrap_content"
      android:text="@string/hello"/>
   <Button
      android:layout_width="100px"
      android:layout_height="wrap_content"
      android:text="Button"
      android:layout_gravity="right"
      android:layout_weight="0.2"/>
   <EditText
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:textSize="18sp"
      android:layout_weight="0.8"/>
</LinearLayout>
```



Android Layout

- The standard Layouts are:
  - AbsoluteLayout
  - FrameLayout
  - LinearLayout
  - RelativeLayout
  - TableLayout

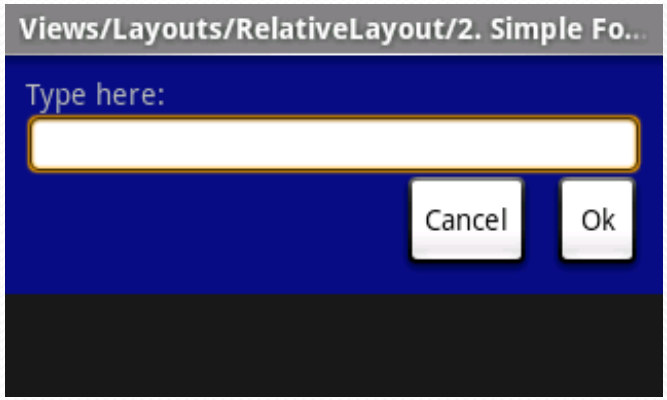
RelativeLayout

- Designed to display child View controls in relation to each other. For instance, you can set a control to be positioned “above” or “below” or “to the left of” or “to the right of” another control, referred to by its unique identifier. You can also align child View controls relative to the parent edges.
- RelativeLayout lets child views specify their position relative to the parent view or to each other (specified by ID)

RelativeLayout

- Example

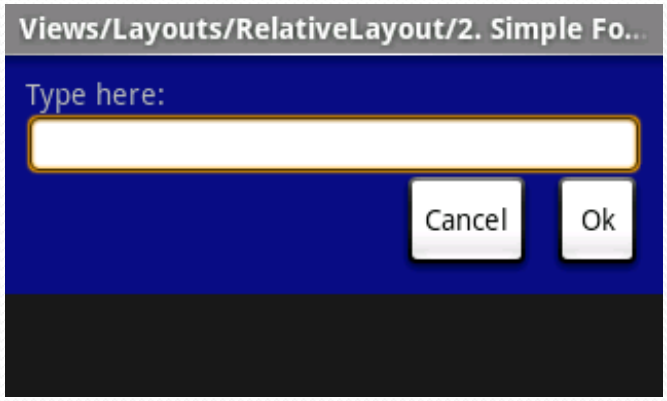
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@drawable/blue"
        android:padding="10px" >
    <TextView android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Type here:" />
    <EditText android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label" />
```



RelativeLayout

- Example

```
<Button android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10px"
        android:text="OK" />
<Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Cancel" />
</RelativeLayout>
```



Android Layout

- The standard Layouts are:
  - AbsoluteLayout
  - FrameLayout
  - LinearLayout
  - RelativeLayout
  - TableLayout

TableLayout

- TableLayout positions its children into rows and columns. TableLayout containers do not display border lines for their rows, columns, or cells. The table will have as many columns as the row with the most cells. A table can leave cells empty, but cells cannot span columns, as they can in HTML
- *TableRow* objects are the child views of a TableLayout (each TableRow defines a single row in the table). Each row has zero or more cells, each of which is defined by any kind of other View. So, the cells of a row may be composed of a variety of View objects, like *ImageView* or *TextView* objects. A cell may also be a ViewGroup object (for example, you can nest another TableLayout as a cell).

TableLayout

- Designed to organize child View controls into rows and columns. Individual View controls are added within each row of the table using a TableRow layout View (which is basically a horizontally oriented LinearLayout) for each row of the table
- TableLayout organizes content into rows and columns. The rows are defined in the layout XML, and the columns are determined automatically by Android. This is done by creating at least one column for each element. So, for example, if you had a row with two elements and a row with five elements then you would have a layout with two rows and five columns.

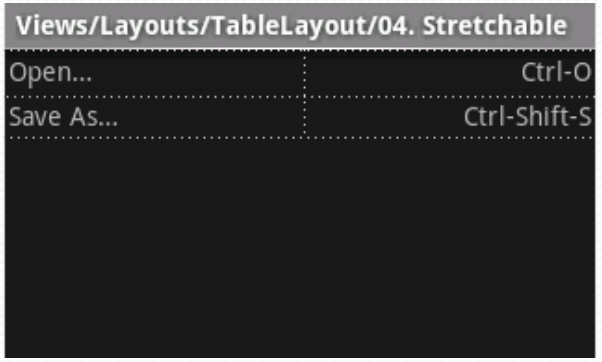
TableLayout

- You can specify that an element should occupy more than one column using android:layout_span. This can increase the total column count as well, so if we have a row with two elements and each element has android:layout_span=”3″ then you will have at least six columns in your table.
- By default, Android places each element in the first unused column in the row. You can, however, specify the column an element should occupy using android:layout_column.

TableLayout

- Example

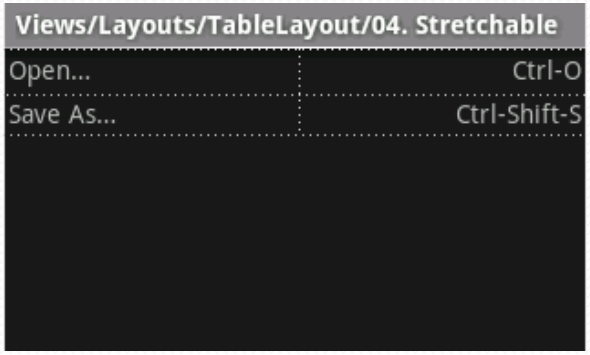
```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:stretchColumns="1">
    <TableRow>
        <TextView
            android:text="@string/table_layout_4_open"
            android:padding="3dip" />
        <TextView
            android:text="@string/table_layout_4_open_shortcut"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
```



TableLayout

- Example

```
<TableRow>
    <TextView
        android:text="@string/table_layout_4_save"
        android:padding="3dip" />
    <TextView
        android:text="@string/table_layout_4_save_shortcut"
        android:gravity="right"
        android:padding="3dip" />
</TableRow>
</TableLayout>
```



END