

Mobile Application Development



03 – Resources & Layout

Objectives

- Introduction to Android Resources and Layouts
- To learn how to layout Android UI controls using XML

RESOURCES

Resources

- Resources also allows you to provide alternative resources that support specific device configurations such as different languages or screen sizes
- Important as more Android-powered devices become available with different configurations
- Examples
 - ID
 - Strings
 - Colors
 - Bitmaps
- Instead of hard coding, use IDs instead
- Lets you change content without changing the code
- E.g.
 - the res/ directory contains all the resources (in subdirectories)
 - an image resource,
 - two layout resources,
 - mipmap/ directories for launcher icons,
 - a string resource file.

```
MyProject/
  src/
    MainActivity.java
  res/
    drawable/
      graphic.png
    layout/
      main.xml
      info.xml
    mipmap/
      icon.png
    values/
      strings.xml
```

Source : <https://developer.android.com/guide/topics/resources/providing-resources.html>

Resource directories

Directory	
color/	XML files that define a state list of colors
drawable/	Bitmap files (.png, .9.png, .jpg, .gif) or XML files that are compiled into the following drawable resource subtypes:Bitmap filesNine-Patches (re-sizable bitmaps)State listsShapesAnimation drawablesOther drawables
layout/	XML files that define a user interface layout.
menu/	XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu.
values/	XML files that contain simple values, such as strings, integers, and colors.

Source : <https://developer.android.com/guide/topics/resources/providing-resources.html>

Resource directories

Directory	
raw/	Arbitrary files to save in their raw form.
xml/	Arbitrary XML files that can be read at runtime by calling <code>Resources.getXML()</code> .
animator/	XML files that define property animations.
anim/	XML files that define tween animations. (Property animations can also be saved in this directory, but the animator/ directory is preferred for property animations to distinguish between the two types.)
mipmap/	Drawable files for different launcher icon densities.

ID

- XML resource that provides a unique identifier for application resources and components.
- A unique resource ID defined in XML
- Android developer tools create a unique integer in your project's R.java class, which you can use as an identifier for an application resources (for example, a View in your UI layout) or a unique integer for use in your application code (for example, as an ID for a dialog or a result code).
- Can be referenced from Kotlin or XML code

- E.g.

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
        android:text="My TextView"
        android:id="@+id/myTV"
/>
```

String Resource

- A string resource provides text strings for your application with optional text styling and formatting.
- There are three types of resources that can provide your application with strings:
 - String
 - String Array
 - Quantity Strings (Plurals)
- Reside in /res/values subdirectory
- File name as **strings.xml**
- **Name** & **Value** pair method
- E.g.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">hello</string>
    <string name="app_name">hello appname</string>
</resources>
```


Name


Value

Color Resources

Localize colors and apply themes

Located at res/values/colors.xml

E.g.

In **colors.xml**

```
<resources>
  <color name="opaque_red">#f00</color>
  <color name="translucent_red">#80ff0000</color>
</resources>
```

In **main.xml**

```
<TextView
  android:layout_width="fill_parent"  android:layout_height="wrap_content"
  android:text="Hello"
  android:textColor="@color/translucent_red "
```

```
/>
```

(Different File)

Compiled / Non-Compiled Resource

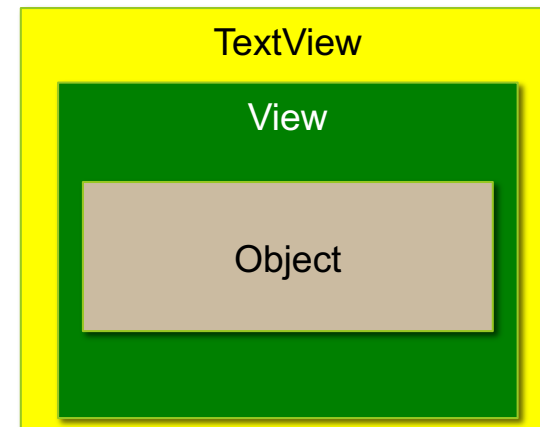
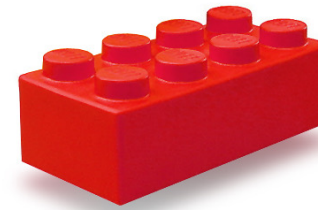
- So far all the xml examples are compiled resources
 - Gets compiled into binary format before becoming part of the installable package
 - Use Android supplied XML readers to read the XML nodes

- Non-Compiled resource
 - Placed in res/raw/ directory
 - Gets copied as-is to the device
 - Explicit stream-based APIs to read these files
 - E.g. Audio and Video files

VIEWS

View

- Building blocks of Android User Interface (UI).
- Added using XML or Kotlin code
- Example: **TextView** widget
 - All widgets inherits from the View class.



TextView

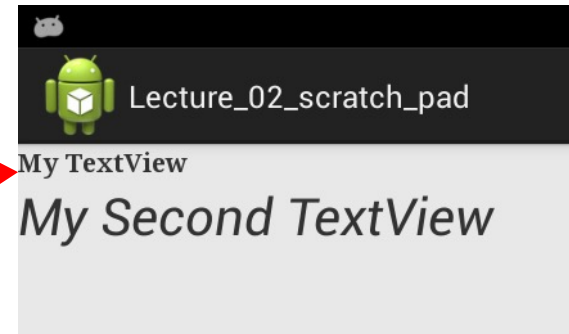
Name
of
class

<TextView

```

android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textStyle="bold"
android:fontWeight="serif"
android:textSize="20px"
android:text="My TextView"
/>

```



Size of Widget

fill_parent

wrap_content

px is one pixel. scale-independent pixels (sp)
and density-independent pixels (dip)

Font Style

Bold

Italics

Normal

Font Type

Sans

Serif

Monospace

Display Text Size

Text to be displayed

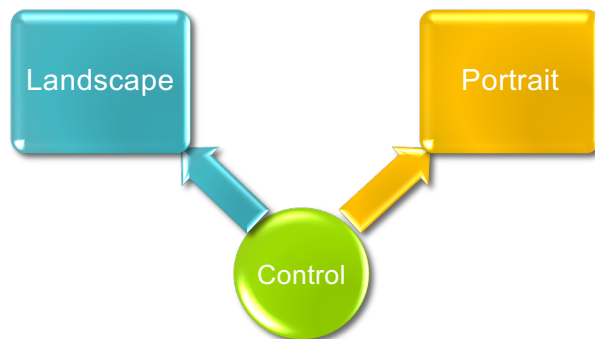
LAYOUTS

Layout

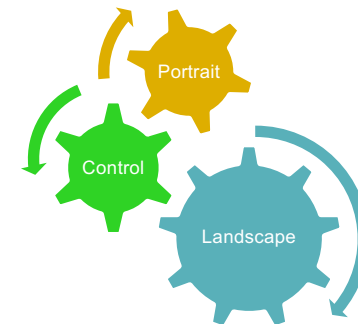
- Defines how widgets (e.g. button, checkbox) are placed and arranged on screen.
- Defined in xml layout resource files.
 - res/layout/

Layout by XML

- Why ?
 - Separate the presentation from the controls
 - Provides the ability to have different variation of UI but only 1 single control
 - Easier to visualize the entire UI



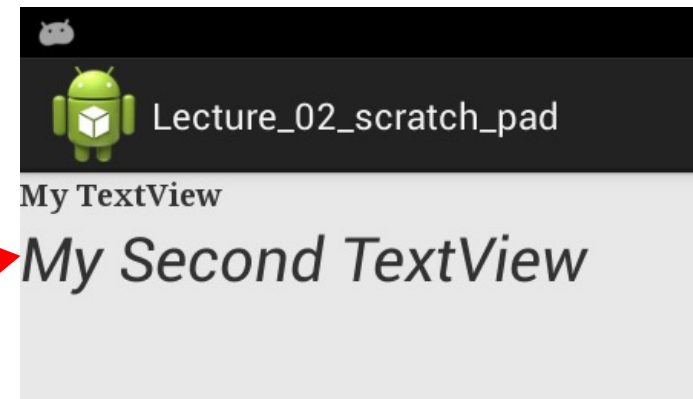
Usage XML to separate controls from UI



Interlocking of logic with UI

Example of an Android Layout XML File

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="fill_parent" android:layout_height="wrap_content"
        android:textStyle="bold"    android:typeface="serif"
        android:textSize="20px"    android:text="My TextView"
    />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textStyle="italic"
        android:typeface="sans"
        android:textSize="40px"
        android:text="My Second TextView"
    />
</LinearLayout>
```



Layout Resources

- A layout resource defines the architecture for the UI in an Activity or a component of a UI.
- In a Kotlin class, the following method is called to load the layout resource for an Activity
 - E.g. “setContentView(R.layout.activity_main)”
- Calls out a constant called “activity_main”
- “main” points to a view defined by an XML layout-resource file
- Expects a file to be called activity_main.xml in “res/layout/”.
 - i.e. “res/layout/activity_main.xml”
- If you have two files under /res/layout/ called file1.xml and file2.xml, you’ll have the following entries in R.java:

```
public static final class layout {
    public static final int file1=0x7f030000;
    public static final int file2=0x7f030001;
}
```

setContentView

- **setContentView**
- So what is the setContentView method doing? It **inflates the layout**. Essentially what happens is that Android reads your XML file and generates objects for each of the tags in your layout file. You can then edit these objects in the codes by calling methods of the objects. We'll go over what these methods look like and how to access view in your layout files later in the course.

Resource Directories

- **Default Resources**
 - Used regardless of the device configuration
 - When no alternative resources match the current configuration
 - E.g. Res/layout/
- **Alternate Resources**
 - Designed for use with specific configuration
 - Append an appropriate configuration qualifier to the directory name
 - E.g. Res/layout-land

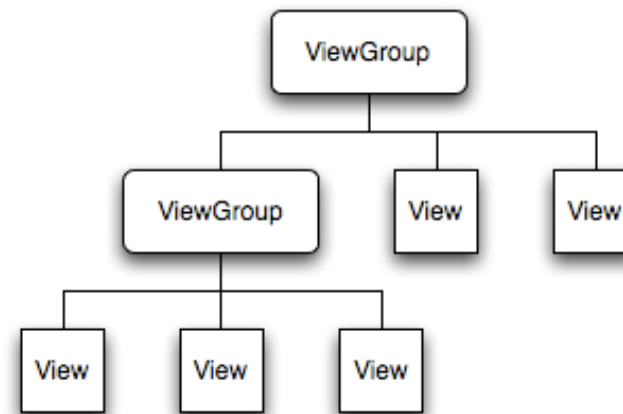


ViewGroup

- Base class for layouts containers.
- Defines layout properties.
- Containers of views and other view groups.
- Added by XML or code.^a

Layouts

- Arranged in a single tree.
- Contains variety of **Views** and **ViewGroups**.
- Arrangement of widgets.
- Assortment of layout types
- Mix and match to suite how your UI should look.



Example - XML

Name
of
class

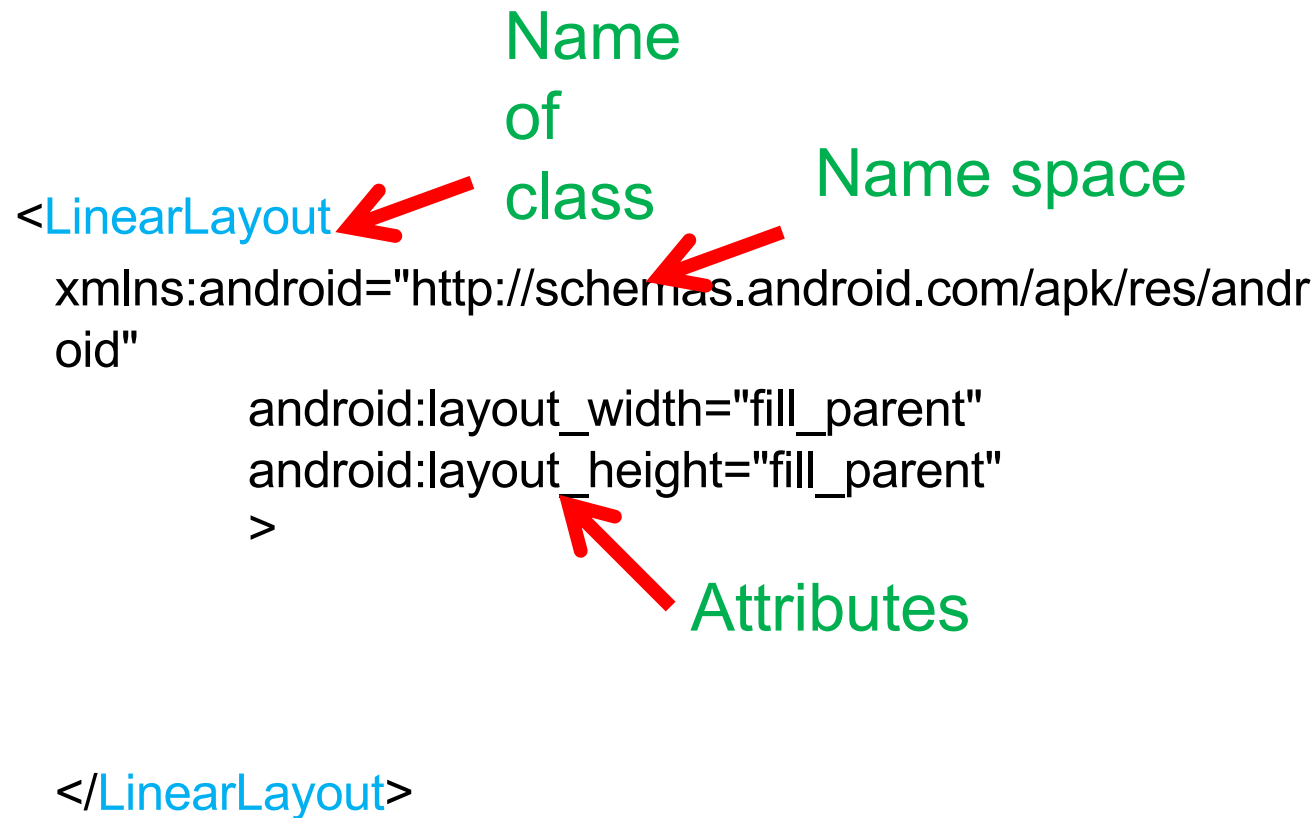
Name space

```

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

</LinearLayout>
  
```

Attributes



Example - XML

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />

    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />

</LinearLayout>

```

Diagram illustrating the XML structure:

- The **LinearLayout** tag is the **Container / Root**.
- The **TextView** and **Button** tags are **Child Item**s.

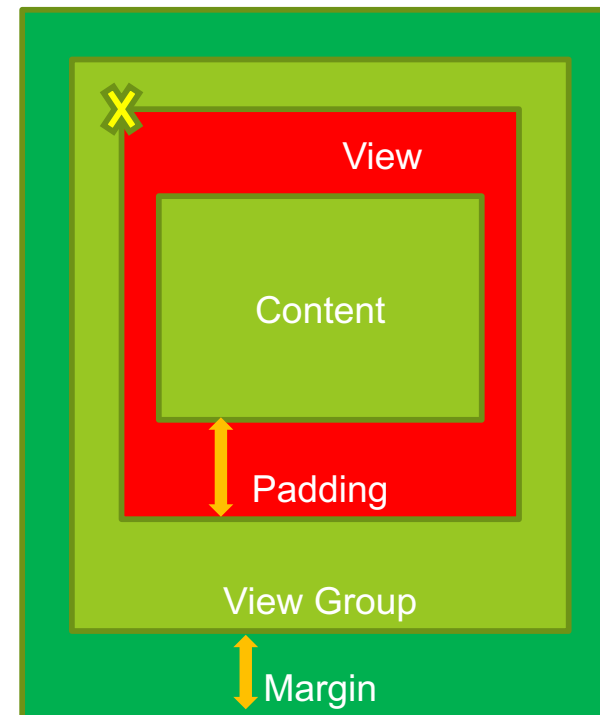
1 x Container with 2 x Child Views

XML Layout Attributes

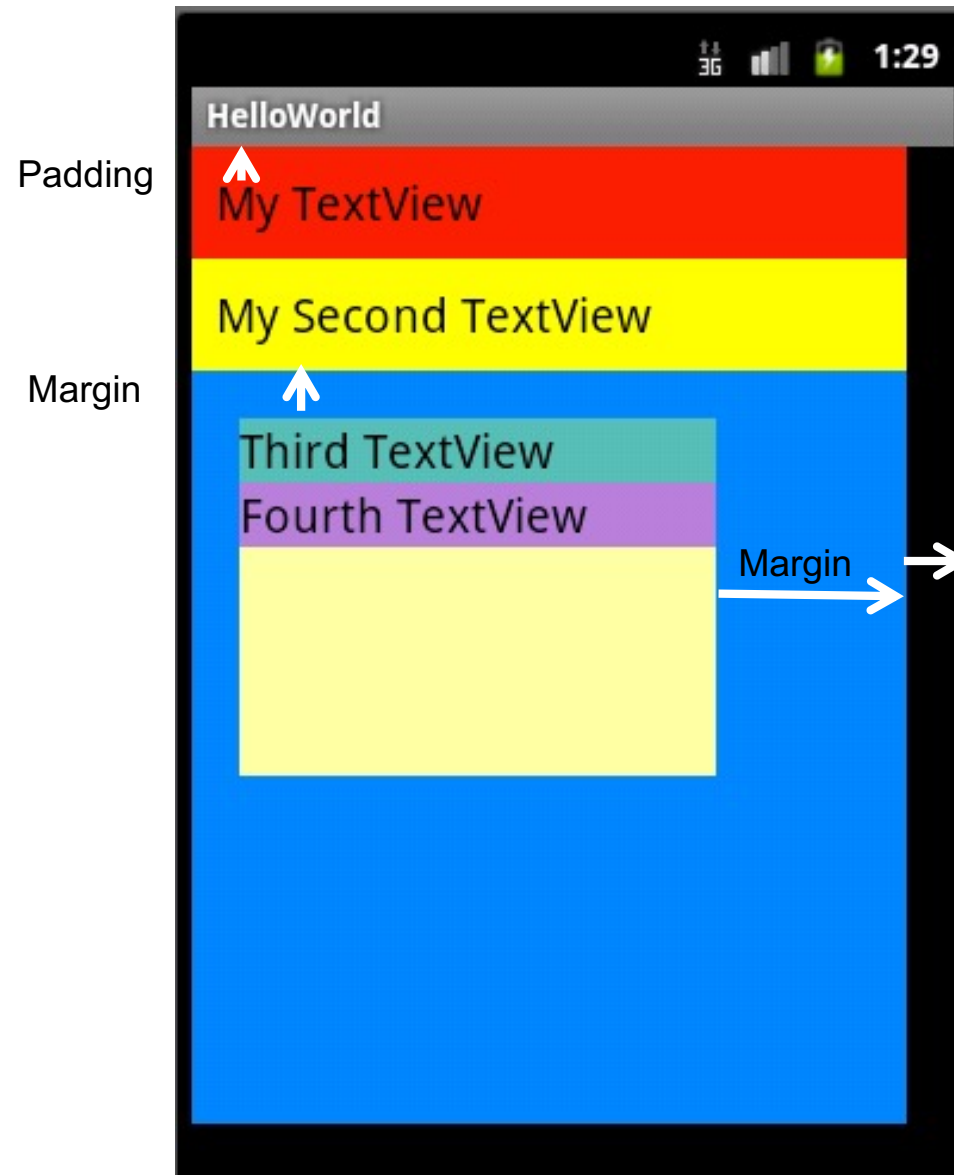
- Layout Parameters
- Size, Padding and Margins
- Specific Attributes

XML Attributes – Layout Sizing

- **Size**
 - Height and width
 - Measured Size
 - Drawing Size
- **Padding**
 - Space inside of content
- **Margins**
 - Space outside of content

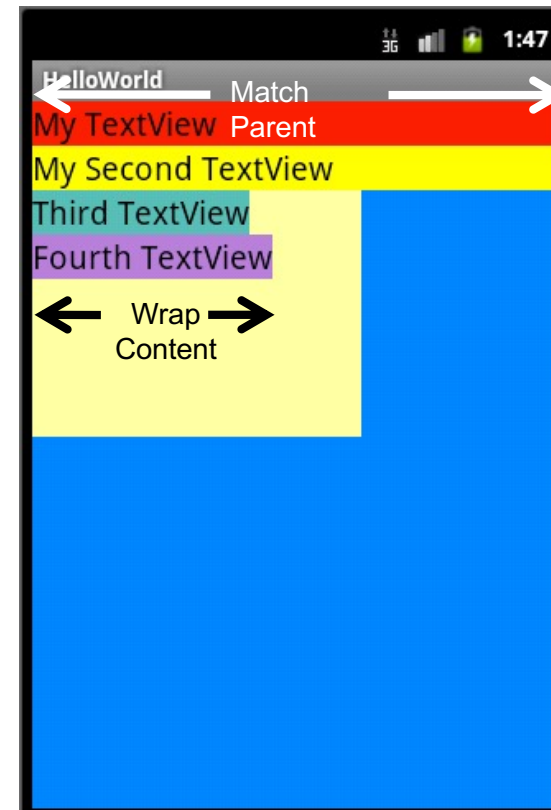


XML Layout Attributes – Layout Sizing



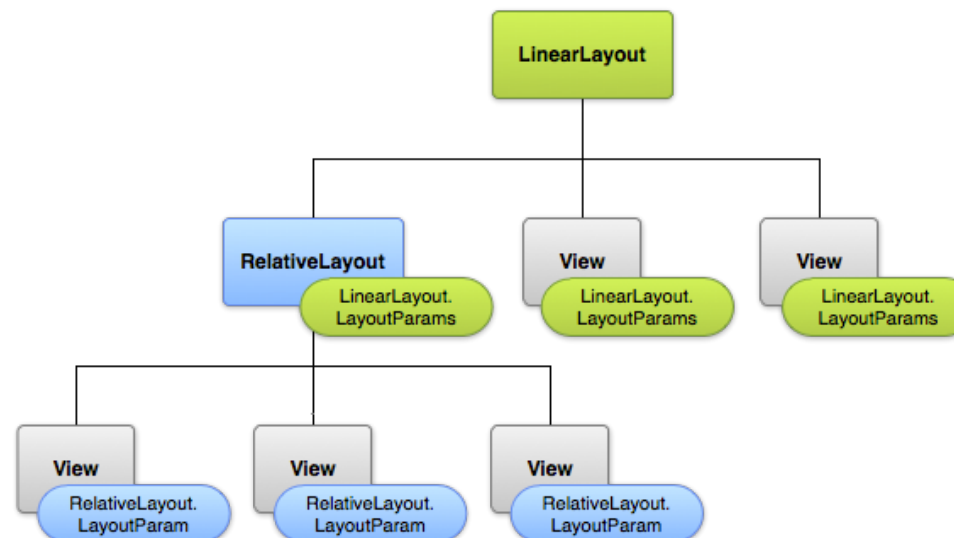
XML Layout Attributes – Layout Sizing

- Match Parent
 - Sets the dimension to match that of the parent element. Added in API Level 8 to deprecate fill_parent.
- Wrap Content
 - Sets the dimension only to the size required to fit the content of this element.



XML Attributes – Layout Parameters

- Layout Params
 - Specify height and width.
 - Parent container define different **LayoutParams** for its own children.
 - Each child element must define **LayoutParams** that are appropriate for its parent.

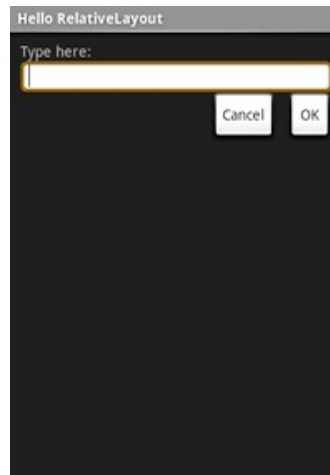


Layout Examples

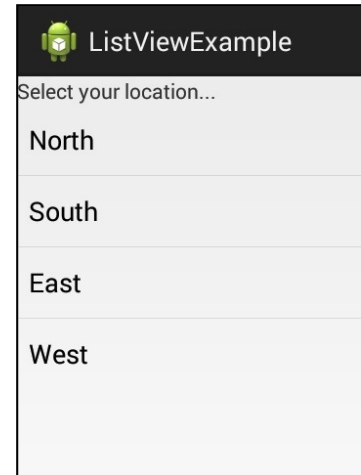
Linear



Relative



ListView



GridView

