UNIVERSIDAD DEL BÍO-BÍO

ICI

VIEWS
INTERFAZ DE USUARIO EN ANDROID

Fernando Santolaya F.
fsantolaya@ubiobio.cl

Resumiendo clases anteriores ….

- ¿Qué hemos trabajado?
  - Creación de aplicaciones básicas con Android
  - Creación de Activity’s
  - Comunicación entre actividades
  - Para lanzar una actividad tenemos los llamados INTENTS
  - EXISTEN 2 TIPOS DE INTENTS
  - INTENTS IMPLICITOS: no se sabe que actividad se lanzará
  - INTENTS EXPLICITOS: se sabe que “actividad” se lanzará
  - Harina de otro costal es el tema de mandar datos entre actividades.
  - Para lo anterior tenemos los Bundle.
  - Podemos también esperar datos de otra actividad.

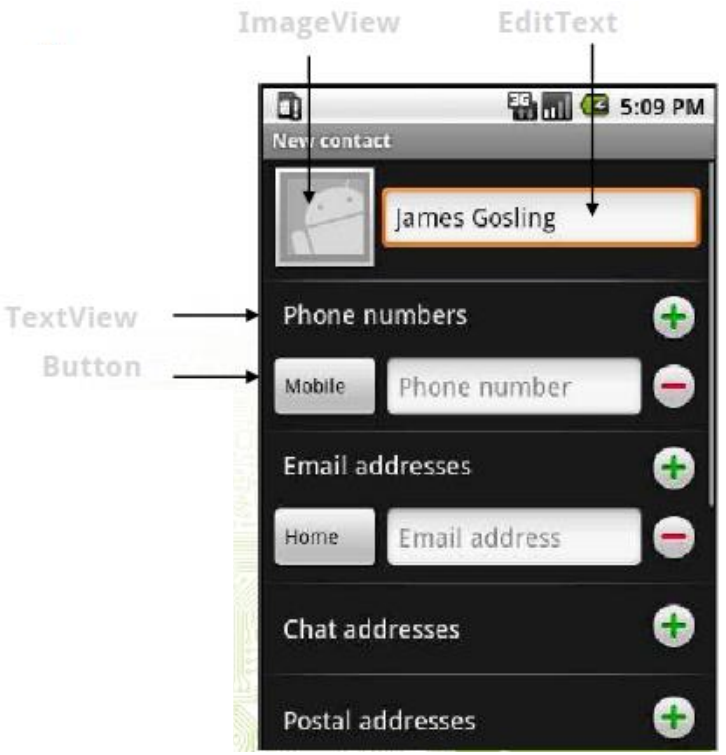

Activities
Views
Intents
Services
Notifications
ContentProviders
AndroidManifest.xml

VIEW

- Una vista es un área rectangular en la pantalla que gestiona el tamaño, el dibujado, el cambio de foco y los gestos del área que representan

- La clase `android.view.View` sirve de clase Base para todos los “widgets”

- Vistas disponibles: `TextView`, `EditText` `Button`, `RadioButton`, `Checkbox`, `DatePicker` `TimePicker`, `Spinner`

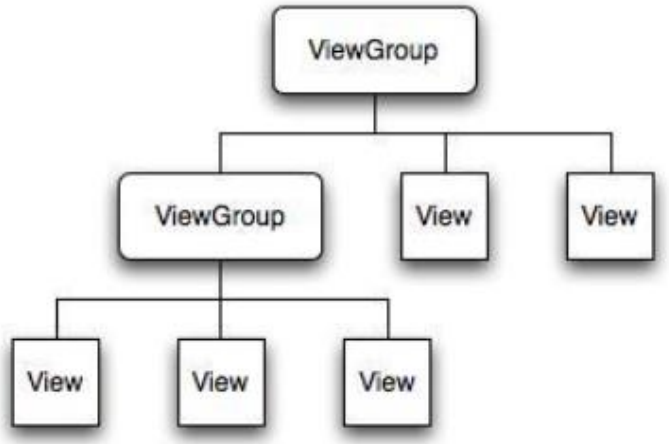


VIEWGROUP

- Una vista especial que contiene otras vistas hijas. ViewGroup es la clase base de los layouts y vistas contenedoras.
- Esta clase también define la clase ViewGroup.LayoutParams
- Layouts: AbsoluteLayout, TableLayout, LinearLayout, RelativeLayout,…
- ViewGroups: DatePicker, Gallery, GridView ListView, ScrollView, Spinner, TabWidget ...

VIEWGROUP

- Las vistas se organizan en estructuras de árbol cuya raíz es un ViewGroup
- setContentView() permite asociar una vista a una actividad.
- La plataforma Android ofrece dos métodos para diseñar la interfaz:
  - **procedural (código)**
  - **declarativa (XML)**




```
public void setContentView(View v);
public void setConentView(View v, LayoutParams p);
public void setConentView(int layoutResID);
```

DISEÑO DE VISTAS

- El framework android permite diseñar la interfaz de manera declarativa en XML. Especificando que se quiere ver en la pantalla y no como se tiene que mostrar. (Similar al HTML)

Procedural – archivo java

```
TextView tv = new TextView(this)
tv.setWidth(100); tv.setHeight(60);
tv.setText("phone");
setContentView(tv);
```

Declarativo – archivo XML

```
<TextView android:id="@+id/nameLabel"
  android:text="phone:"
  android:layout_width="100"
  android:layout_height="60"
/>
```

INTERFAZ DE USUARIO

- El método declarativo permite separar la presentación de la aplicación del código que contrala su comportamiento.

- El tenerlos separarlos permite modificar la interfaz de la aplicación sin modificar el código fuente. Así, se podría diseñar layouts para diferentes orientaciones de la pantalla, diferentes tamaños de pantalla o diferentes idiomas sin tocar el código fuente.

- Existe un convenio de nombres entre los artributos del xml y los métodos de los objetos.

```
<TextView
  android:text="phone:"/>
 ...
TextView tv= new TextView(this)
tv.setText("Phone");
```

INTERFAZ DE USUARIO

- Las vistas heredan atributos a sus clases base y definen sus propios atributos.

- El atributo id identifica a la vista dentro del árbol y permite recuperarla desde la aplicación.

- El símbolo (@) indica al parser del xml que lo que viene a continuación lo trate como un identificador de recurso. El símbolo (+) indica que el nombre que viene a continuación es un nombre nuevo y debe ser añadido a la clase de recursos R.java.

```
<TextView
  android:id="@+id/nameLabel"
  android:text="phone:"
  android:layout_width="100"
  android:layout_height="60"
/>
```

INTERFAZ DE USUARIO

- Cuando se compila la aplicación se compila también cada archivo xml de presentación y queda accesible desde la clase R “View Resource” generada por android.

```
<Button
  android:id="@+id/acceptButton"
  android:text="@string/acceptButton"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
/>
```

- findViewById. Permite acceder a la vista desde una actividad

```
Button btn = (Button) findViewById(R.id.acceptButton);
```

- R.Java Clase generada por android que permite acceder a los recursos una vez compilados como atributos estáticos

LAYOUTS DE ANDROID

LAYOUTS DE ANDROID

- Un layout es un contenedor invisible que determina la disposición de las vistas en la pantalla. Todos los layouts heredan de `ViewGroup`.

- Android recomienda definir el layout en formato XML mediante archivos que se encuentran ubicados en (ejemplo) `res/layout/pantalla.xml`.

- Cuando se compila la aplicación se compila también cada archivo xml de layout y queda accesible mediante la clase `R` generada por android.

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.pantalla);
}
```

TIPOS DE LAYOUTS

- **LinearLayout:** dispone los hijos horizontalmente o verticalmente.
- **RelativeLayout:** dispone cada elemento relativo a los demás .
- **TableLayout:** dispone los elementos en filas y columnas
- **AbsoluteLayout:** dispone los elementos en coordenadas exactas
- **FrameLayout:** permite cambiar dinámicamente los controles en el layout.

- La clase `Layout` contiene una clase `LayoutParams` que es específica por cada tipo de Layout. (Ejemplo: `LinearLayoutParams`, `RelativeLayoutParams`, …)

LINEARLAYOUT

- Es un modelo de caja, los hijos o widgets son alineados arriba en una columna o fila, uno después de otro.

- Para configurar un `LinearLayout` existen 5 áreas de control principales:
  - orientación, modelo de llenado, peso, gravedad, relleno.

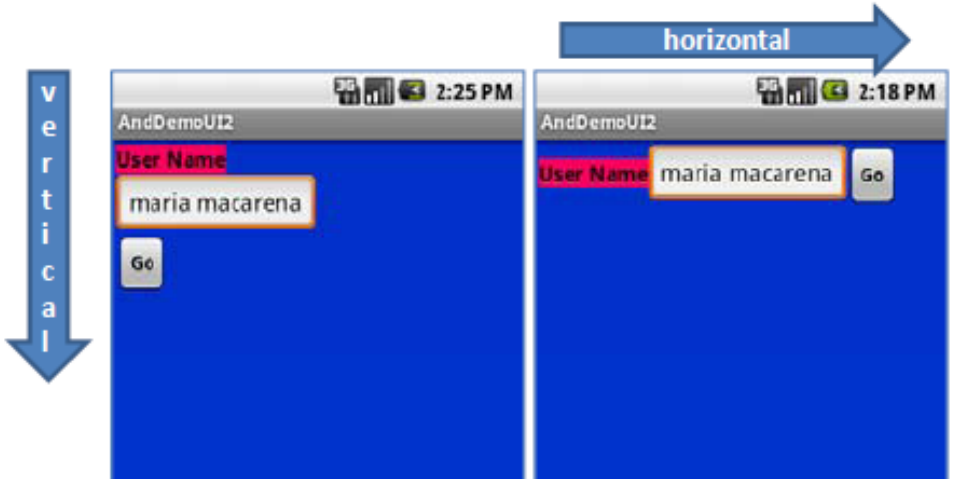
Orientación (Orientation)

- Indica si se debe organizar en filas o columnas.

- Agregar la propiedad `android:orientation` a la etiqueta `<LinearLayout>` en el archivo XML. Sus valores pueden ser "`horizontal`"→filas o "`vertical`"→columnas.

- La orientación puede ser modificada en tiempo de ejecución invocando a `setOrientation()`

LINEARLAYOUT

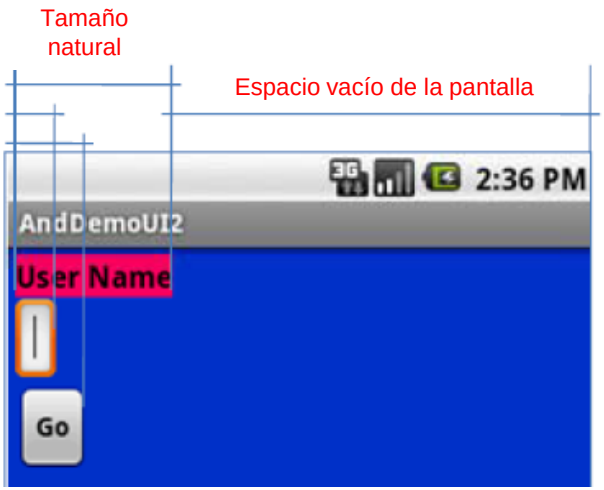



```
<LinearLayout
   ...
   android:orientation="vertical" >
   ...
</LinearLayout>
```

```
<LinearLayout
   ...
   android:orientation="horizontal" >
   ...
</LinearLayout>
```

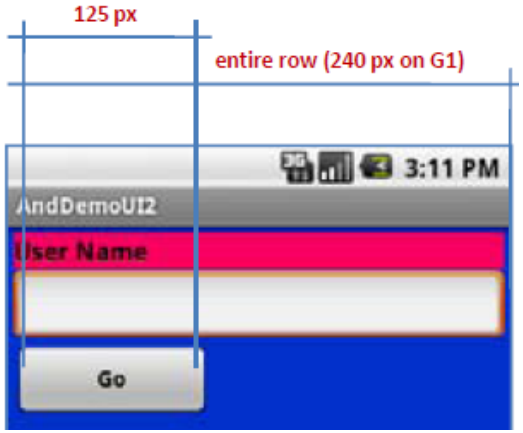
Modelo de llenado (Fill Model)

- Los widget tienen un tamaño natural basado en su texto acompañante.
- Cuando sus tamaños combinados no calzan exactamente con el ancho de la pantalla, debemos determinar qué hacer con el espacio restante.



LINEARLAYOUT

- Todos los widget dentro de un `LinearLayout` deben proveer atributos dimensionales `android:layout_width` y `android:layout_height` para ayudar a direccionar el resto del espacio vacío.
- Los valores usados en la definición de heigth y width son:
  - Especificar una dimensión particular, tal como `125px` para indicar 125 pixeles exactos.
  - Usar `wrap_content`, que significa que el widget puede llenar su espacio natural, a menos que sea muy grande, en cuyo caso Android rellena el espacio necesario.
  - Usar `fill_parent`, que significa que el widget puede ocupar todo el espacio en su contenedor, después que los otros widgets se ubiquen.




```
<LinearLayout
   ...
   android:orientation="vertical" >

   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      ...
   </EditText>
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      ...
   <Button
      android:layout_width="125px"
      android:layout_height="wrap_content"
</LinearLayout>
```

LINEARLAYOUT

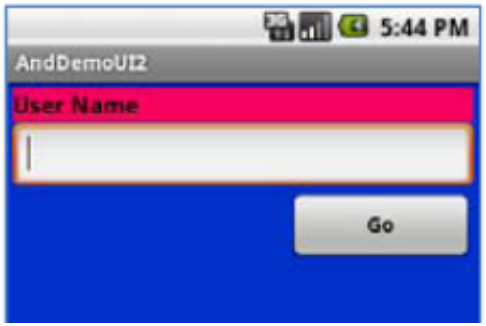
Peso (Weight)

- Es utilizado para asignar espacios **proporcionales** a los widgets de una vista.
- Puede usar `android:layout_weight` con los valores 1, 2, 3, … para indicar que proporción del espacio libre podría tener cada widget.
- Ejemplo: Usamos el mismo ejemplo anterior modificado. A los dos componentes (`TextView` y `Button`) le agregamos propiedades adicionales `android:layout_weight="1"` entonces el control `EditText` tiene `android:layout_weight="2"`.

LINEARLAYOUT

Gravedad (Gravity)

- Es utilizado para alinear el control en la pantalla.
- Por defecto, los widget se alinean a la izquierda y arriba.
- Tú puedes usar la propiedad XML `android:layout_gravity="…"` para establecer otras posibles alternativas: `left`, `center`, `right`, `top`, `bottom`, etc.

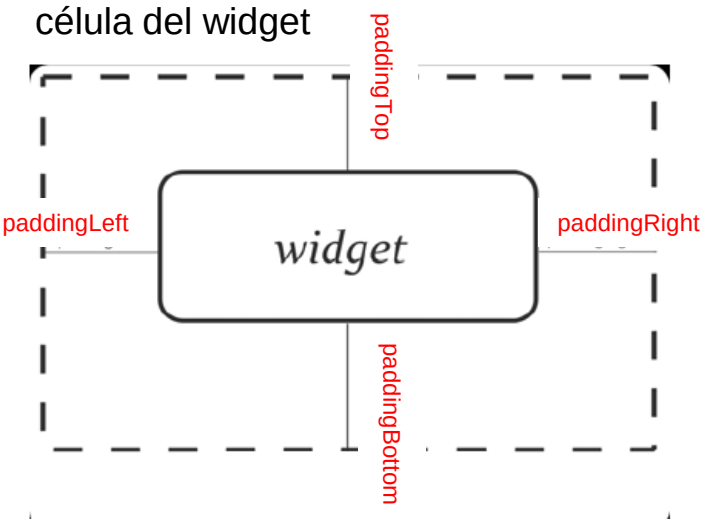


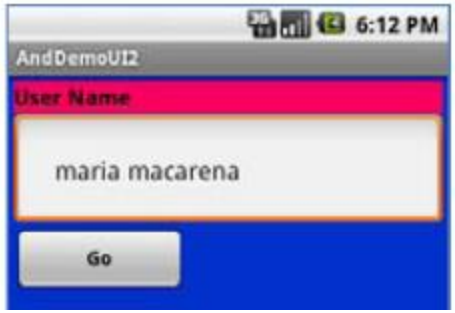
El botón tiene gravedad derecha

LINEARLAYOUT

Relleno (Padding)

- El padding es análogo a los márgenes en procesador de texto como Word.
- Especifica que tanto espacio hay entre los bordes del widget.
- Si desea incrementar el espacio entre widgets se puede utilizar la propiedad `android:padding` en XML o invocar al método `setPadding()` en tiempo de ejecución.






```
<EditText
   android:id="@+id/ediName"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:textSize="18sp"
   android:padding="30px”>
</EditText>
...
```

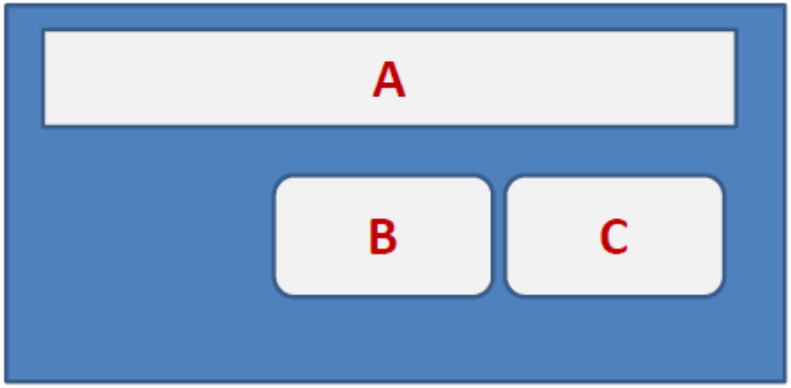
LINEARLAYOUT - EJEMPLO

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  android:id="@+id/myLinearLayout"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="#ff0033cc"
  android:padding="4px"
  android:orientation="vertical"
 xmlns:android="http://schemas.android.com/apk/res/android"
>
<TextView
  android:id="@+id/labelUserName"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:background="#ffff0066"
  android:text="User Name"
  android:textSize="16sp"
  android:textStyle="bold"
  android:textColor="#ff000000”>
</TextView>
```


```
<EditText
  android:id="@+id/ediName"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:textSize="18sp"
  android:layout_margin="10px">
</EditText>
<Button
  android:id="@+id/btnGo"
  android:layout_width="125px"
  android:layout_height="wrap_content"
  android:text="Go"
  android:textStyle="bold"
  android:layout_gravity="right"
  android:layout_marginRight="10px">
</Button>
</LinearLayout>
```

RELATIVELAYOUT

- Ubica los widget basado en su relación con otros widgets en el contenedor actual y contenedor padre.



Ejemplo:

- A está en la parte superior del padre.
- C está debajo de A, a su derecha.
- B está debajo de A, a la izquierda de C.

RELATIVELAYOUT

- Las siguientes propiedades boleanas mapean el widget de acuerdo a su ubicación **respecto de la ubicación del padre**:

  - **`android:layout_alignParentTop`** el tope del widget podría alinerse con el tope del contenedor.
  - **`android:layout_alignParentBottom`** la base del widget podría alinearse con la base del contenedor.
  - **`android:layout_alignParentLeft`** el lado izquierdo del widget podría alinearse con el lado izquierdo del contenedor.
  - **`android:layout_alignParentRight`** el lado derecho del widget podría alinearse con el lado derecho del contenedor.
  - **`android:layout_centerHorizontal`** el widget podría ser ubicado horizontalmente al centro del contenedor.
  - **`android:layout_centerVertical`** el widget podría ser posicionado verticalmente al centro del contenedor.
  - **`android:layout_centerInParent`** el widget podría ser posicionado horizontalmente y verticalmente al centro del contenedor.

RELATIVELAYOUT

- Las siguientes propiedades boleanas mapean el widget de acuerdo a su ubicación **respecto de la ubicación de otros widget**:

  - `android:layout_alignParentTop` indica que el tope del widget podría alinerse con el tope del contenedor.
  - `android:layout_above` indica que el widget podría ser ubicado arriba del widget referenciado en la propiedad.
  - `android:layout_below` indica que el widget podría ser ubicado abajo del widget referenciado en la propiedad.
  - `android:layout_toLeftOf` indica que el widget podría ser ubicado a la izquierda del widget referenciado en la propiedad.
  - `android:layout_toRightOf` indica que el widget podría ser ubicado a la derecha del widget referenciado en la propiedad.

RELATIVELAYOUT

- `android:layout_alignTop` indica que el tope del widget podría estar alineado con el tope del widget referenciado en la propiedad.
- `android:layout_alignBottom` indica que la base del widget podría estar alineada con la base del widget referenciado en la propiedad.
- `android:layout_alignLeft` indica que el lado izquierdo del widget podría estar alineado con el lado izquierdo del widget referenciado en la propiedad.
- `android:layout_alignRight` indica que el lado derecho del widget podría estar alineado con el lado derecho del widget referenciado en la propiedad.
- `android:layout_alignBaseline` indica que las líneas base de los dos widget podrían estar alineadas.

RELATIVELAYOUT

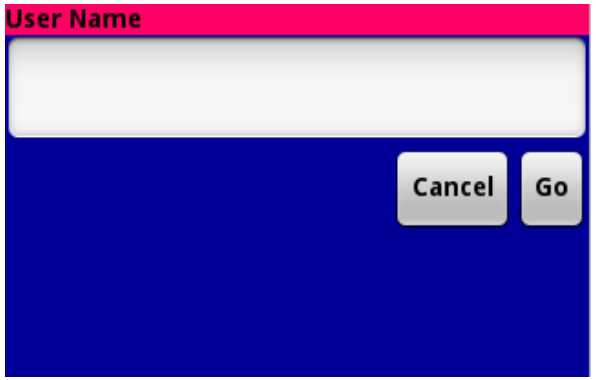
- Para usar la notación relativa en las propiedades se necesita mantener la consistencia:

1. Poner identificadores (atributos `android:id`) en todos los elementos que necesites direccionar. La sintáxis es: `@+id/…` (por ejemplo, un `EditText` podría ser referenciado así: `android:id="@+id/ediUserName"`).

2. Referencia otros widgets usando el mismo valor del identificador (`@+id/…`) otorgado a ese widget. Por ejemplo, un control bajo el `EditText` podría decir: `android:layout_below="@+id/ediUserName"`.

RELATIVELAYOUT - EJEMPLO

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  android:id="@+id/myRelativeLayout"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="#ff000099"
  xmlns:android="http://schemas.android.com/apk/res/android">
<TextView
  android:id="@+id/lblUserName"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:background="#ffff0066"
  android:text="User Name"
  android:textStyle="bold"
  android:textColor="#ff000000"
  android:layout_alignParentTop="true"
  android:layout_alignParentLeft="true">
</TextView>
<EditText
  android:id="@+id/ediUserName"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:layout_below="@+id/lblUserName"
  android:layout_alignParentLeft="true"
  android:layout_alignLeft="@+id/myRelativeLayout"
  android:padding="20px">
</EditText>
```

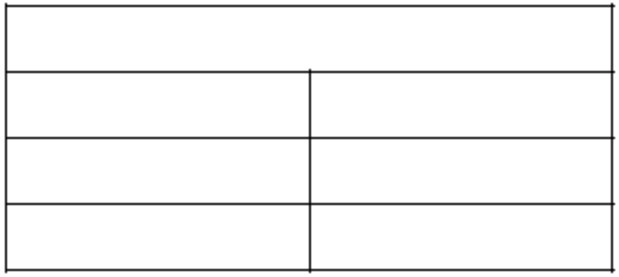



```
<Button
  android:id="@+id/btnGo"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_below="@+id/ediUserName"

android:layout_alignRight="@+id/ediUserNa
me"
  android:text="Go"
  android:textStyle="bold">
</Button>
<Button
  android:id="@+id/btnCancel"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_toLeftOf="@+id/btnGo"
  android:layout_below="@+id/ediUserName"
  android:text="Cancel"
  android:textStyle="bold">
</Button>
</RelativeLayout>
```

TABLELAYOUT

- Permite indicar la posición de los widget en base a una grilla de filas y columnas.



- Las columnas pueden encogerse (shrink) o estirarse (stretch) para acomodar sus contenidos.

- `TableLayout` funciona en conjunto con `TableRow`.

- `TableLayout` controla todo el comportamiento del contenedor, con los widgets posicionados en uno o más contendores `TableRow`, uno por fila de la grilla.

TABLELAYOUT

- Las filas se delaran colocando widget como hijos de una `TableRow`, todo dentro de un `TableLayout`.

- El número de columnas es determinado por Android (nosotros controlamos el número de columnas de manera indirecta).

- Ejemplo:

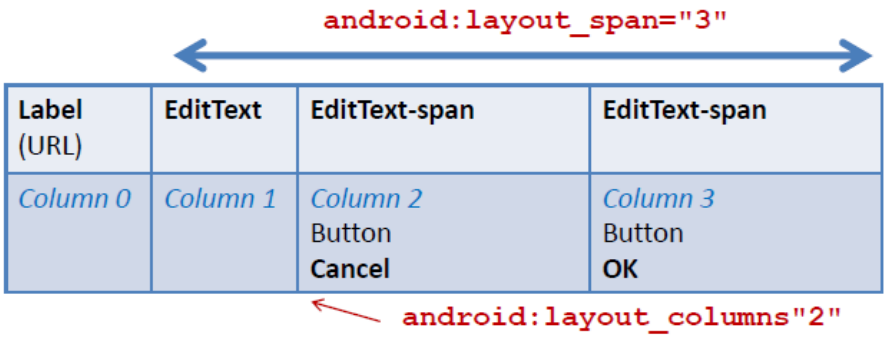
0	1	2	3

TABLELAYOUT

- Sin embargo, un widget puede ocupar más de una columna incluyendo la propiedad `android:layout_span`, que indica el número de columnas que el widget abarca.

```
<TableRow>
  <TextView android:text="URL:" />
  <EditText
     android:id="@+id/entry"
     android:layout_span="3" />
</TableRow>
```

- Normalmente, los widget son ubicados en la primera columna de cada fila. Agregando la etiqueta "URL" ubicamos la primera columna (la columna 0). En el ejemplo anterior, el `TextField` podría estar dentro de un conjunto de 3 columnas mezcladas (span) . Columna 1 a 3.



TABLELAYOUT - EJEMPLO

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
  android:id="@+id/myTableLayout"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="#ff0033cc"
  android:orientation="vertical"
  xmlns:android="http://schemas.android.com/apk/res/android">
  <TableRow>
    <TextView
      android:text="URL:" />
      <EditText android:id="@+id/ediUrl"
      android:layout_margin="10px"
      android:layout_span="3"/>
  </TableRow>
  <View
    android:layout_height="3px"
    android:background="#0000FF"
  />
```


```
<TableRow>
  <Button android:id="@+id/cancel"
  android:layout_column="2"
  android:text="Cancel" />
  <Button android:id="@+id/ok"
  android:text="OK" />
</TableRow>
<View
  android:layout_height="3px"
  android:background="#0000FF" />
</TableLayout>
```

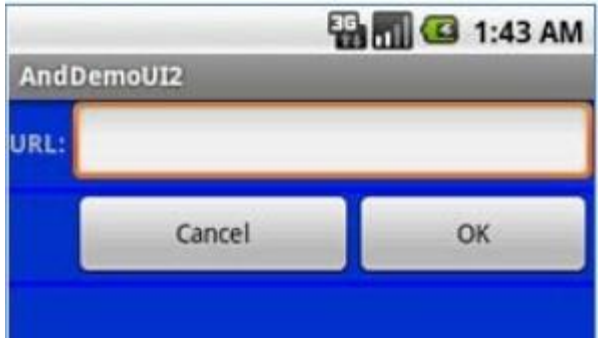
TABLELAYOUT

- Por defecto, cada columna tendrá un tamaño “natural” de acuerdo al tamaño de los widget que estén en ella.

- Si el contenido es más estrecho que el espacio disponible, se puede usar la propiedad: android:stretchColumns=“…”

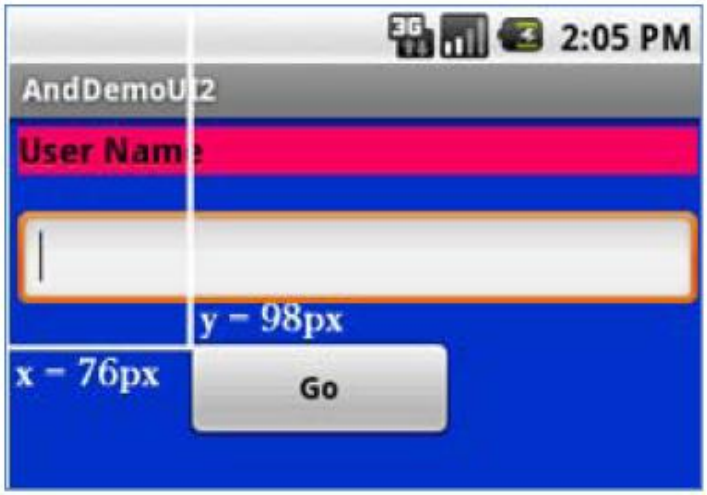
- Su valor podría ser el número de una sola columna o una lista de columnas separas por comas. Estas columnas serán expandidas para ocupar cualquier espacio disponible sobre la fila.

```
...
<TableLayout
  android:id="@+id/myTableLayout"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="#ff0033cc“
  android:orientation="vertical"
  android:stretchColumns ="2,3,4"
  xmlns:android="http://schemas.android.com/apk/res/android"
>
...
```



ABSOLUTELAYOUT

- Permite especificar la ubicación exacta de sus hijos en coordenadas x / y.
- Este tipo de contenedor es menos flexible y difícil de mantener que los otros tipos de layouts.



ABSOLUTELAYOUT - EJEMPLO

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
  android:id="@+id/myLinearLayout"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="#ff0033cc"
  android:padding="4px"
  xmlns:android="http://schemas.android.com/apk/res/android"
>
<TextView
  android:id="@+id/labelUserName"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:background="#ffff0066"
  android:text="User Name"
  android:textSize="16sp"
  android:textStyle="bold"
  android:textColor="#ff000000"
  android:layout_x="0px"
  android:layout_y="-1px”>
</TextView>
```


```
<EditText
  android:id="@+id/ediName"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:textSize="18sp"
  android:layout_x="0px"
  android:layout_y="38px"
>
</EditText>
<Button
  android:id="@+id/btnGo"
  android:layout_width="125px"
  android:layout_height="wrap_content"
  android:text="Go"
  android:textStyle="bold"
  android:layout_y="98px"
  android:layout_x="46px">
</Button>
</AbsoluteLayout>
```

SCROLLVIEW

- Cuando tenemos muchos datos para mostrar sobre una sola pantalla podemos usar el control `ScrollView`.
- Este control provee un sliding o acceso con scroll a los datos. Esto es similar a ver una página web muy larga.
- Es posible poner cualquier control dentro del `ScrollView`, por ejemplo listas de `TextView` e `ImageView`, o realizar grandes formularios.

SCROLLVIEW - EJEMPLO

```
<?xml version="1.0" encoding="utf-8"?>
<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:text="Un formulario muy largo"
      android:textSize="20dp"
      android:textColor="#ED0034"/>
   <!-- inicia el ScrollView -->
   <ScrollView android:layout_width="fill_parent" android:layout_height="fill_parent">
      <!-- se colocal el LinearLayout que contendra el formulario -->
      <LinearLayout
          android:orientation="vertical"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent">
          <TextView android:text="Nombre"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"/>
          <EditText android:layout_width="fill_parent"
                    android:layout_height="wrap_content"/>
          <TextView android:text="Apellido"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"/>
          <EditText android:layout_width="fill_parent"
                    android:layout_height="wrap_content"/>
          ...
      </LinearLayout>
   </ScrollView>
</LinearLayout>
```

Unidades de Medida

Existen varias maneras de especificar el tamaño de los elementos de la interfaz gráfica.

- **px** is one pixel.
- **sp** is scale-independent pixels.
- **dip** is Density-independent pixels.

Density Bucket	Screen Density	Physical Size	Pixel Size
ldpi	120 dpi	0.5 x 0.5 in	0.5 in * 120 dpi = 60 x 60 px
mdpi	160 dpi	0.5 x 0.5 in	0.5 in * 160 dpi = 80 x 80 px
hdpi	240 dpi	0.5 x 0.5 in	0.5 in * 240 dpi = 120 x 120 px
xhdpi	320 dpi	0.5 x 0.5 in	0.5 in * 320 dpi = 160 x 160 px
xxhdpi	480 dpi	0.5 x 0.5 in	0.5 in * 480 dpi = 240 x 240 px

You would use

- `sp` for font sizes
- `dip` for everything else.

dip==dp

Tarea

Resolvieron el tema de los QR? Y la captura de imágenes ?