Interfaz de usuario II
Layouts y controles gráficos

Curso: Desarrollando aplicaciones con Android

Contenido

- Interfaz de usuario
- Layout
- Vistas
- Adaptadores
- Eventos de interacción
- Estilos y temas

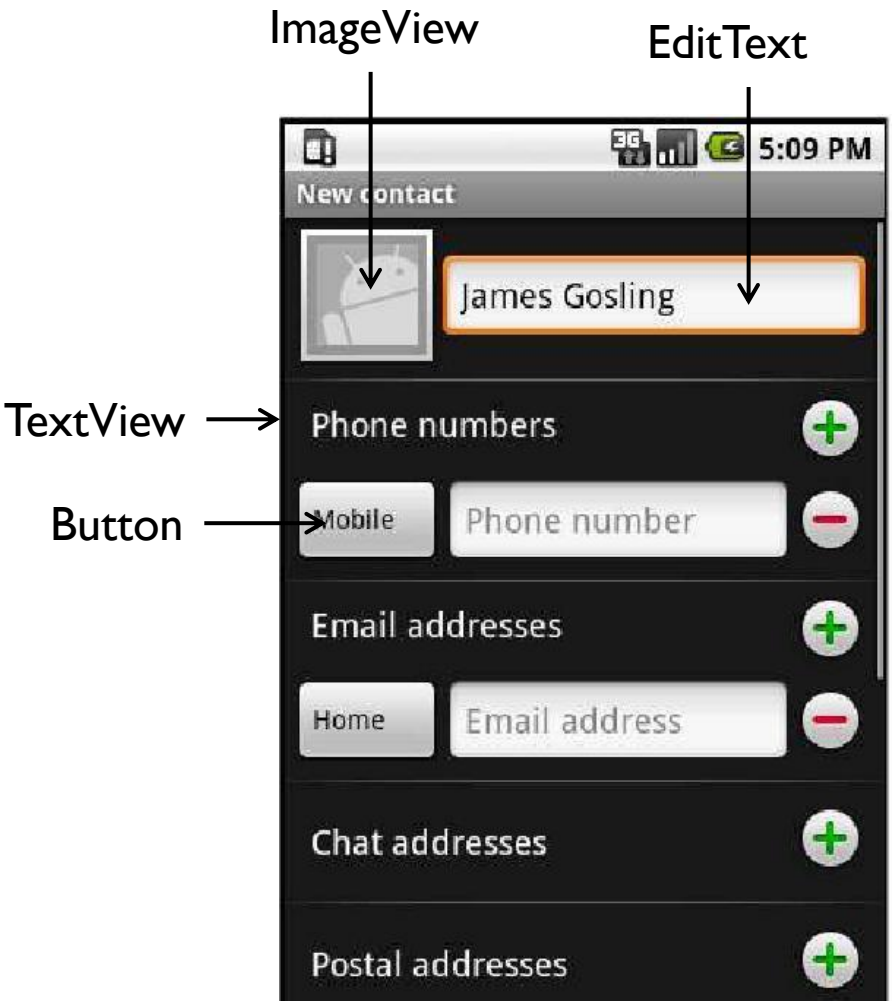
Interfaz de usuario

- **Activity** [Form]
  - Pantalla que se muestra al usuario. Las actividades se componen de vistas
- **View** [Component]
  - Controles o widgets de la interfaz de usuario. Elemento básico de la interfaz que permite la interacción con el usuario
- **ViewGroup** [Container]
  - Composición de vistas. Los controles complejos y el Layout heredan de la clase base ViewGroup.

"Existen muchas vistas y layout's predefinidos aunque puedes crear los tuyos propios"

Interfaz de usuario

- **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

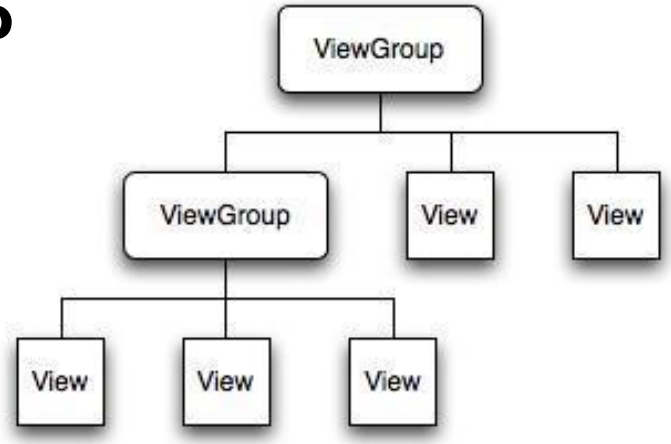


Interfaz de usuario

- ViewGroup
  - Una vista especial que contiene otras vistas hijas.
  - ViewGroup es la clase base 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 ...

Interfaz de usuario

- Las vistas se organizan en estructuras de árbol cuya raíz es un **ViewGroup**
- **setContentView()** permite añadir 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) {}
```

Interfaz de usuario

- **Diseñando interface de forma declarativa XML**
  - 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 Java

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

Interfaz de usuario

- **Diseñando interface de forma declarativa XML**
  - El método declarativo permite separar la presentación de la aplicación del código que contrala su comportamiento.
  - El tenerlos separados 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 **atributos del xml** y los **métodos de los objetos**.

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

Interfaz de usuario

- **Diseñando interface de forma declarativa XML**
  - Las vistas heredan los atributos de 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

- Diseñando Interface de forma declarativa XML
  - 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

Layout

- Los Layout's son contenedores invisibles que determinan la disposición de las vistas en la pantalla.
- Todos los layouts heredan de ViewGroup
- Android recomienda definir el layout en **XML** mediante archivos de layout que se encuentran en **res/layout/**
- Cuando se compila la aplicación se compila también cada archivo xml de layout y queda accesible desde la **clase R** "View Resource" generada por Android.
- Asignar un layout a una actividad:

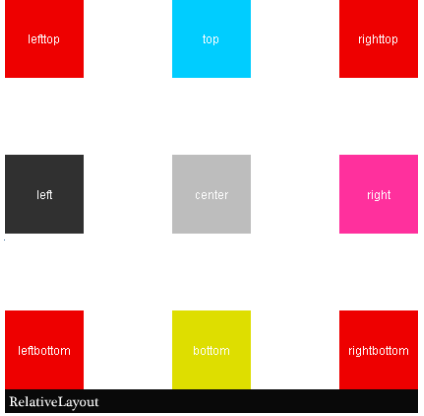
```
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main); }
```

Layout

- Tipos de layout:
  - **LinearLayout**: dispone los hijos horizontal o verticalmente
  - **RelativeLayout**: dispone unos elementos relativos a los otros
  - **TableLayout**: dispone los elementos en filas y columnas
  - **AbsoluteLayout**: dispone los elementos en coordenadas exactas
  - **FrameLayout:** Pensado para solo un view, se muestran como una pila

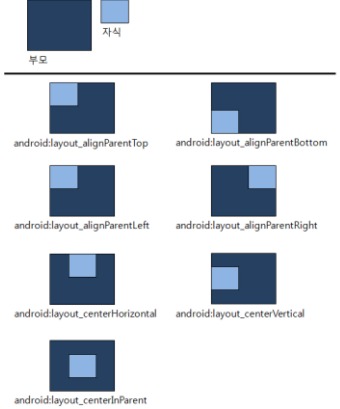
- La clase Layout contiene una clase **LayoutParams** que es especializada por cada tipo de Layout. (LinearLayoutParams, RelativeLayoutParams,…)

Layout



- Propiedades específicas del RelativeLayout
  - Posición relativa a otro control:
    - layout_above posiciona el borde inferior de la vista por encima de la indicada
    - layout_below posiciona el borde superior de la vista por debajo de la indicada
    - layout_toLeftOf posiciona el borde derecho de la vista a la izquierda de la indicada
    - layout_toRightOf posiciona el borde izquierdo de la vista a la derecha de la indicada
    - layout_alignLeft posiciona el borde izquierdo de la vista en el borde izquierdo de la indicada
    - layout_alignRight posiciona el borde derecho de la vista en el borde derecho de la indicada
    - layout_alignTop posiciona el borde superior de la vista en el borde superior de la indicada
    - layout_alignBottom posiciona el borde inferior de la vista en el borde inferior de la indicada
    - layout_alignBaseline posiciona la línea base del texto de la vista coincidiendo con la indicada. Ver el siguiente ejemplo:

Layout



- Propiedades específicas del RelativeLayout
  - Posición relativa al *Layout* padre:
    - layout_alignParentLeft posiciona el borde izquierdo de la vista en el borde izquierdo del Layout.
    - layout_alignParentRight posiciona el borde derecho de la vista en el borde derecho del Layout.
    - layout_alignParentTop posiciona el borde superior de la vista en el borde superior del Layout.
    - layout_alignParentBottom posiciona el borde inferior de la vista en el borde inferior del Layout.
    - layout_centerHorizontal centra la vista horizontalmente en relación al Layout.
    - layout_centerVertical centra la vista verticalmente en relación al Layout.
    - layout_centerInParent centra la vista horizontalmente y verticalmente en relación al Layout.

Vistas

- Android viene con un conjunto de vistas "controles" bastante completo para dibujar textos, botónes, obtener fechas, mostrar listas, mostrar imágenes, webs, mapas, etc.
- **Texto**
  - Android incluye distintos componentes para mostrar y permitir la entrada de texto por parte del usuario.
  - **TextView** Muestra el texto. No permite editarlo (Label)
  - **EditText** Componente de edición de texto, acepta varias líneas
  - **AutoCompleteTextView** Ofrece sugerencias del texto escrito
  - **MultiAutoCompleteTextView** Ofrece sugerencias de cada palabra

Vistas

- **Botones**
  - Existen todo tipo de botones, botones simples, con imágenes, check buttons, radio buttons, toogle buttons…
  - **Button** Botón estandar
  - **ImageButton** Botón con imagen
  - **ToogleButton** Botón de 2 estados On | Off
  - **CheckBox** Checkbox estandar
  - **RadioButton** RadioButton estándar. Selección simple

Vistas

- **ListView**
  - Muestra una lista de items verticalmente. Se puede diseñar la apariencia de los elementos de la lista. Normalmente se usa con una ListActivity y un ListAdapter.
  - Pasos para crear un ListView
    1. Diseñar la vista de los elementos de la lista en res/layout/listItem.xml
    2. Crear una actividad que herede de ListActivity
    3. Pasarle los datos al adaptador de la lista
    4. En onCreate asignarle el ListAdapter a la actividad

Vistas

- ListView

1

```
<TextView android:id="@+id/listItem"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent"/>
```

2

```
public class HelloListView extends ListActivity {
   private String[] fruits = {"Orange", "Apple", "Pear"};
   private ArrayAdapter[] adp = null;
   private ListView lv1;
   public void onCreate(Bundle savedInstance) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.MainActivity);
      lv1 =(ListView)findViewById(R.id.listView1);
3     adp = new ArrayAdapter(this,
            android.R.layout.simple_list_item_1, fruits);
4     lv1.setAdapter(adp);

   }
}
```

Vistas

- **WebView**
  - Permite embeber un navegador en una actividad. El navegador incluye opciones para navegar hacia adelante y hacia atrás, zoom, etc
  - Pasos para crear un WebView
    1. Diseñar la vista en res/layout/main.xml
    2. Modificar el AndroidManifest.xml para añadir permisos y configurar a pantalla completa
    3. Crear una actividad
    4. En onCreate llamar al setContentView pasándole el id de la vista

Vistas

- WebView

1
```
<WebView android:id="@+id/webview"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"/>
```

2
```
En AndroidManifest.xml
<uses-permission android:name="android.permission.INTERNET" />
<activity android:name=".webView"
         android:label="@string/app_name
         android:theme="@android:style/Theme.NoTitleBar">
```

3
4
```
public void onCreate(Bundle savedInstance) {
   super.onCreate(icicle);
   setContentView(R.layout.main);
   webView = (WebView) findViewById(R.id.webview);
   webView.loadUrl("http://www.google.es");
}
```

Vistas

- Atributos más importantes de las vistas
  - Posición de la vista dentro del layout
    - layout_width, layout_height Permite ajustar el ancho y alto de la vista. Se puede indicar una dimensión concreta, por ejemplo 200px, aunque lo habitual es utilizar uno de los valores:
      - wrap_content ajusta el tamaño a las dimensiones necesarias para representar el contenido.
      - fill_parent ajusta el tamaño al máximo posible según el *Layout* padre que la contiene. Ha sido renombrado match_parent a partir del nivel de API 8, aunque podemos utilizar también el nombre anterior.
    - layout_margin, layout_margin_botton, layout_margin_left, layout_margin_right, layout_margin_top Establece un margen exterior a la vista.
    - layout_gravity Centra o justifica la vista dentro del *Layout*.
    - layout_weight Cuando estamos en un LinealLayout y se dispone de espacio libre sin utilizar, podemos repartirlo entre las vistas del Layout de forma que este se reparte proporcionalmente al valor indicado en este parámetro.

Vistas

- Atributos más importantes de las vistas
  - Definición del comportamiento
    - id Define el identificador que nos permitirá acceder a la vista. Para crear nuevos identificadores utilizar la expresión "@+id/nombre_identificador". El carácter @ significa que se trata de un identificador de recurso (es decir se definirá en el fichero *R.java*). El carácter + significa que el recurso ha de ser creado en este momento. También existen ciertos identificadores que ya han sido definidos en el sistema. Por ejemplo, más adelante utilizaremos "@android:id/list"para crear un ListView.
    - tag Permite almacenar un String que podrá ser utilizado para cualquier fin. Es decir, una información extra que el programador podrá usar para fines específicos.
    - content_description Cadena de caracteres que describe el contenido de la vista.
    - clickable Indica si la vista reacciona ante eventos de tipo onClick (se pulsa sobre la vista).
    - on_click Nombre del método que será invocado cuando ocurra un evento onClick (a partir de la versión 1.6).
    - long_clickable Indica si la vista reacciona a eventos de tipo pulsación larga (más de un segundo).
    - focusable Indica si la vista puede tomar el foco.
    - focusable_in_touch_mode Establece que cuando el dispositivo tenga capacidades de pantalla táctil y se pulsa sobre la vista esta tomará el foco. Hay que diferenciarlo de clickable. Por ejemplo, nos suele interesar que un botón pueda recibir evento *onClick* pero no que coja el foco.
    - next_focus_down, next_focus_left, next_focus_right, next_focus_up Permite especificar el movimiento del foco cuando usamos las cuatro teclas de cursor. En la mayoría de los casos no hace falta indicarlo, ya que se ajustará automáticamente según la posición de las vistas.

Vistas

- Atributos más importantes de las vistas
  - Aspectos visuales
    - visibility Permite hacer invisible una vista
    - visible la vista es visible
    - invisible la vista es invisible pero ocupa lugar
    - gone la vista es invisible pero no ocupa lugar
    - background Permite establecer una imagen de fondo.
    - style Permite aplicar un estilo a la vista. (Ver apartado estilos y temas.)
    - min_width, min_height Ancho y alto mínimo de la vista.
    - padding, paddingBottom, paddingTop, paddingLeft, paddingRigh Establece un margeninterior en la vista. Tiene sentido en vistas como *Button* para establecer un margen entre el texto y el borde del botón. Por el contrario, layout_margin establece la separación entre el borde del botón y otras vistas.

Vistas

- Atributos más importantes de las vistas
  - Atributos para texto
    - text Texto que se mostrará
    - text_size Tamaño del texto
    - text_style Estilo del texto (negrita ó itálica)
    - typefaceTipo de fuente usada en el texto
    - gravity Cómo el texto es alineado dentro de la vista
    - text_appearance Permite definir conjuntamente el tipo de fuente, tamaño del texto, color,…
    - text_color Color del texto
    - text_color_link Color del texto para hipervínculos.
    - text_color_highlight Color del texto cuando es seleccionado
    - text_color_hint Color del texto de indicación (ver hint).
    - text_scale_x Deforma el texto con un factor de escala horizontal.
    - width, height Hace que el texto tenga exactamente el ancho o alto especificado
    - hint Texto que se mostrará, normalmente dentro de un EditText, aunque en otro color para indicar algún tipo de instrucciones. Por ejemplo “Introduzca aquí su nombre”.

Adaptadores

- Un adaptador es un objeto que convierte la interfaz de una clase o otra que el cliente espera. En Android se usan los adaptadores para adaptar los datos a otro formato para que se puedan mostrar en una vista.
- Existen diferentes tipos de adaptadores:
  - ListAdapter
  - ArrayAdapter
  - SpinnerAdapter
  - SimpleCursorAdapter
- También se pueden definir adaptadores propios.

Adaptadores

- **Ejemplo de uso de un adaptador**

```
public void onCreate(Bundle savedInstance) {
   adapter = new SimpleCursorAdapter( this,
                  R.layout.main_item_two_line_row,
                  cursor,
                  new String[] { TITLE, TEXT },
                  new int[] { R.id.text1, R.id.text2 });
   setListAdapter(adapter);
}
```

Parameters:

- Context context
- int layout id
- Cursor cursor
- String[] from
- int[] to

Eventos de interacción

- Normalmente la interacción del usuario con el sistema es modelada con **eventos.** Si una aplicación quiere enterarse y responder a una interacción del usuario ha de añadir la lógica apropiada para detectar y procesar el evento.
- Un evento encapsula la información necesaria para que el manejador pueda tratar esa entrada.
- Android trata los eventos mediante:
  - **Event Handlers**
  - **Event Listeners**
- **Event Handler:** maneja los eventos de entrada sin importar donde está el foco. No están necesariamente asociados a una vista. Ej: pulsar el Botón atrás, tocar la pantalla
- **Event Listener:** escuchan eventos generados por una View o ViewGroup. Cada Event Listener tiene solo un método callback, que será llamado por el framework Android cuando el usuario interactúa con la vista. Ej onClick, onLongClick, onFocusChanged,…

Eventos de interacción

- **Event Handler** maneja el evento sin importar quien tiene el foco. Una actividad define los event Handlers más comunes. ListActivity y un Dialog tienen los suyos propios
- **Event Handler callbacks en una actividad**
  - onKeyUp Se libera una tecla
  - onKeyDown Se pulsa una tecla
  - onTouchEvent Se toca la pantalla
  - onTrackballEvent Se apreta/mueve el trackball
  - onBackPressed Se pulsa el botón atrás

Eventos de interacción

- Un **EventListener** escucha eventos generados por una vista. Previamente es necesario registrarlo mediante el método **setOnXXXListener** apropiado.

```
public class EventActivity extends Activity
                           implements OnClickListener {
   private View view;
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      view = (View) findViewById(R.id.view);
      view.setOnClickListener(this);
   }
}
```

- Android recomienda que la actividad implemente la interfaz onXXXListener en lugar de usar clases anónimas

Estilos y temas

- Se sigue la misma filosofía que la CSS en el diseño web.
- En Android el estilo se especifica en un archivo XML

Contenido y estilo

```
<TextView android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:textColor="#00FF00"
          android:typeface="monospace"
          android:text="@string/hello" />
```

Estilo separado

```
<TextView style="@style/CodeFont"
          android:text="@string/hello" />
```

- El **estilo** es una apariencia que se aplica a una vista
- El **tema** es un estilo que se aplica a una activida o a una aplicación

Estilos y temas

- **Definiendo un estilo**
  - Para crear un conjunto de estilos se crear un archivo xml en **res/values** cuya raiz sea el tag <resources>. Para cada estilo que se quiera crear se añade un tag <style>.

```
<resources>
   <style name="CodeFont">
      <item name="android:layout_width">fill_parent</item>
      <item name="android:layout_height">wrap_content</item>
      <item name="android:textColor">#00FF00</item>
   </style>
</resources>
```

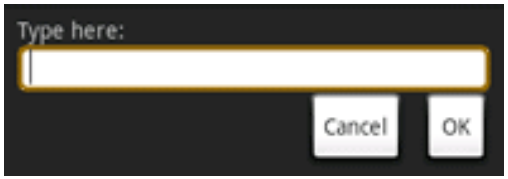
  - Cada hijo de resources es convertido en un recurso de la aplicación al compilar y puede ser **referenciado desde un XML como @style/CodeFont**
  - Podemos usar "Extract style" para crear un estilo a partir de uno creado

Ejercicio 2.7 – Creación visual de vistas

- Construye visualmente la siguiente interfaz de usuario
  - Linear layout
  - Toggle button
  - Checkbox (10 px margen, centrado)
  - Progressbar
  - RatingBar
- Tiene que parecerse a la imagen
- no ser exactamente igual

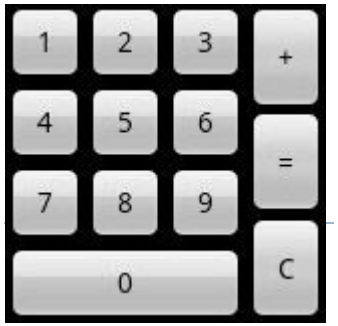
Ejercicio 2.8 – Uso de layouts

1. Utiliza un RelativeLayout para realizar un diseño similar al siguiente:



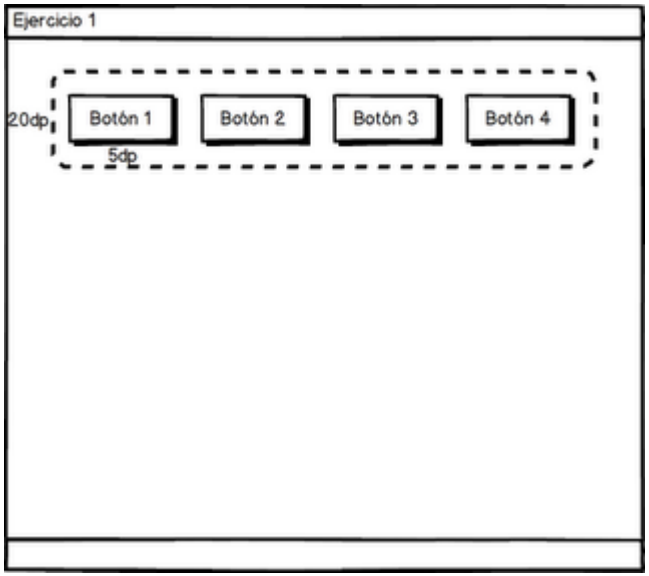
2. Utiliza un TableLayout para realizar un diseño similar al siguiente:

3. Utiliza un AbsoluteLayout para realizar un diseño similar al siguiente. Ojo AbsoluteLayout lo encontrarás en la paleta *Advanced* no en la de *Layouts*. Trata de reutilizar el *Layout* creado en el punto anterior.



Ejercicio 2.9 – Uso de layouts

- Fila de cuatro botones horizontalmente ocupando todo el ancho de la pantalla.
- Cada botón tiene que ocupar el mismo espacio, o sea, el 25% del ancho de la pantalla. (Al cada botón android:layout_weight="1")
- Dejar un espacio alrededor del grupo de botones de 20dp.
- Separar entre sí cada uno de los botones por 5dp.



Ejercicio 2.10 – Uso de layouts

- Construir el siguiente formulario:

Menú de opciones

- En la carpeta menu (Contiene los items a mostrar)

```
<menu
  xmlns:android="http://schemas.android.com/apk/
  res/android" >

     <item
         android:id="@+id/menu_settings"
         android:orderInCategory="100"
         android:title="@string/menu_settings"/>
     <item android:id="@+id/item1"></item>

</menu>
```

Menú de opciones

- El método OnCreateOptionsMenu crea el menú de opciones.
- R.menu.main es el archivo que se encuentran las diferentes opciones de menú.

```
public boolean onCreateOptionsMenu(Menu menu) {
// Inflate the menu; this adds items to the action bar if
   it is present.
getMenuInflater().inflate(R.menu.main, menu);
return true;
}
```

Menú de opciones

- Para capturar los eventos dentro del método onOptionsItemSelected.
- A través de item.getItemId() podemos saber que opción se ha pulsado.

```
public boolean onOptionsItemSelected(MenuItem item) {
        // TODO Auto-generated method stub
        if (item.getItemId()==R.id.AcercaDe){

….
}
}
```

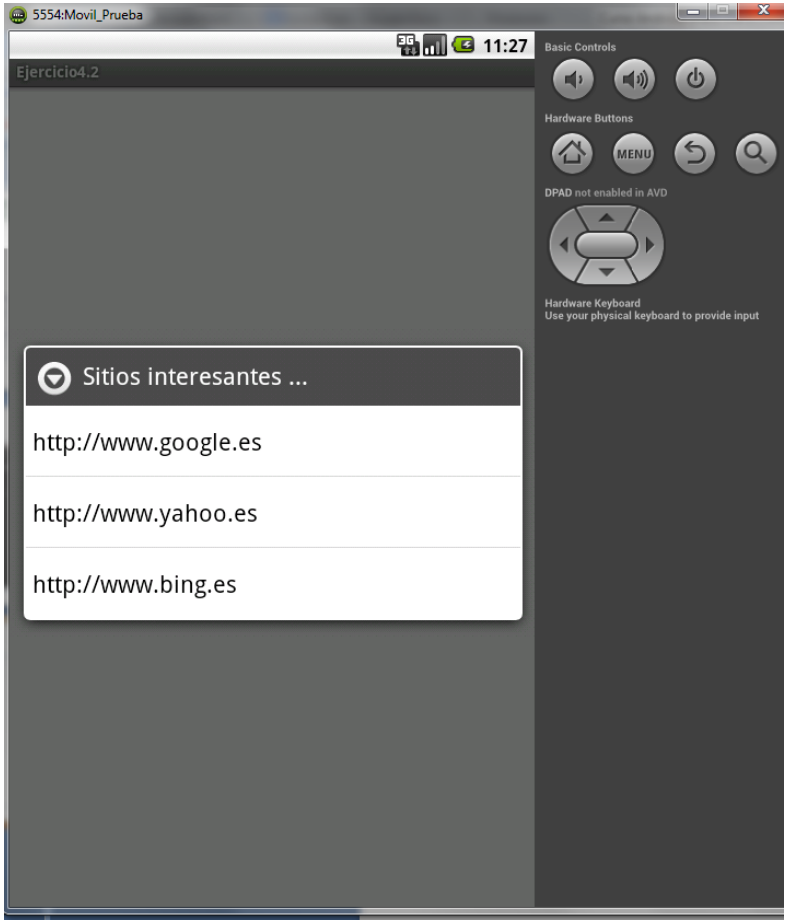
Ejercicio 2.11a – Menu acerca de..

Crear un menú de opciones que muestre el autor en un Toast en "Acerca de" y la opción de salir.

Toast.*makeText*(***this, "Autor: J. Enrique Agudo",*** Toast.*LENGTH_LONG*)***.show();***

Ejercicio 2.11b – Menú sitios interesantes..

Añadir un menú de opciones que muestre un submenú con enlaces a sitios Web y la opción de salir.



Sub menús

Menu.xml

```
..
 <item android:id="@+id/sitios" android:title="@string/Sitios">
        <menu>
            <item android:id="@+id/google" android:title="@string/google"/>
            <item android:id="@+id/yahoo" android:title="@string/yahoo"/>
        </menu>
    </item>
…
```

En .java

```
public boolean onOptionsItemSelected(MenuItem item) {
…

if (item.getItemId()==R.id.google) {
Intent i = new Intent("android.intent.action.VIEW",Uri.parse("http://www.google.es"));
 startActivity(i);
…
}
```

Ejercicio 2.11c – Menu contextual

Añadir un menú contextual que modifique el color de un EditText

Menu contextual

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   et1 = (EditText)findViewById(R.id.editText1);
   registerForContextMenu(et1);
}


@Override
public boolean onContextItemSelected(MenuItem item) {
// TODO Auto-generated method stub
   if (item.getItemId()==R.id.rojo) {
   et1.setBackgroundColor(Color.rgb(255,0,0));
   }
   else if (item.getItemId()==R.id.verde) {
   et1.setBackgroundColor(Color.rgb(0,255,0));
   }
   return super.onContextItemSelected(item);
}
```