CONTENIDO

1. DISEÑO DE LA INTERFAZ DE USUARIO 2

1.1. Creación de una interfaz de usuario por código 2

1.2. Creación de una interfaz de usuario usando XML 2

1.2.1. Edición visual de las vistas 4

1.2.2. Layouts 5

1.2.2.1. Ejercicio Resuelto 1 (Código fuente en la carpeta EjercicioGridLayout) 8

1.2.2.2. Ejercicio 1 9

1.2.3. Controles gráficos 10

1.2.3.1. Botones 10

1.2.3.2. Imágenes, etiquetas y cuadros de texto 12

1.2.3.3. Texto con formato en controles TextView y EditText 13

1.2.3.4. Checkbox y RadioButton 13

1.2.3.5. Listas desplegables 14

1.2.3.5.1. Adaptadores en Android (adapters) 14

1.2.3.5.2. Control Spinner 15

2. APLICACIÓN DE EJEMPLO: SOLOBICI DIDÁCTICO 17

2.1. Ejercicios 18

2.1.1. Ejercicio Resuelto 2 18

2.1.2. Ejercicio 2 19

2.1.3. Ejercicio Resuelto 3 20

1. DISEÑO DE LA INTERFAZ DE USUARIO

El diseño de la interfaz de usuario en Android sigue una filosofía muy diferente a la de otras plataformas. En Android la interfaz de usuario no se diseña exclusivamente con código Java, sino utilizando XML o incluso utilizando una interfaz gráfica.

1.1. Creación de una interfaz de usuario por código

Vamos a ver un primer ejemplo de cómo crear una interfaz de usuario utilizando exclusivamente código. Esta no es la forma recomendada de trabajar con Android pero nos será de ayuda para entender algunos conceptos.

Si necesitamos introducir un campo de texto en una aplicación como por ejemplo el HolaCefireCheste realizado en el tema anterior, el archivo *HolaCefireChesteActivity.java* quedaría de la siguiente forma:

```
package android.aguilar;

import android.app.Activity;

public class HolaCefireChesteActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.main);
        TextView texto = new TextView(this);
        texto.setText("Hola Alumnos del Cefire de Cheste");
        setContentView(texto);
    }
}
```

Sin embargo esta forma de crear las interfaces de usuario no es la correcta ya que el mantenimiento de la misma y los posibles cambios serían muy costosos. Como bien sabemos todos, un principio importante en el diseño de software es que conviene separar todo lo posible el diseño, los datos y la lógica de la aplicación.

1.2. Creación de una interfaz de usuario usando XML

Android proporciona una alternativa para el diseño de interfaces de usuario. Se trata de los ficheros de diseño basados en XML. Vamos a ver uno de estos ficheros. Para ello accede al fichero res/layout/main.xml de uno de nuestros proyectos. El siguiente ejemplo proporcionaría un resultado similar que el ejemplo de diseño por código anterior.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

</LinearLayout>
```

Resulta sencillo interpretar su significado. Se introduce un elemento de tipo LinearLayout. Su función es contener otros elementos de tipo View. Este LinearLayout tiene cuatro atributos. El primero, xmlns:android, es una declaración de un espacio de nombres de XML que utilizaremos en Android (Este parámetro solo es necesario especificarlo en el primer elemento). El atributo android:orientation indica que los elementos se van a distribuir de forma vertical. Los otros dos permiten definir el ancho y alto de la vista. En el ejemplo se ocupará todo el espacio disponible en la pantalla puesto que hemos utilizado los valores fill_parent. También tenemos el valor wrap_content para que el tamaño dependa del valor que haya dentro del elemento.

Dentro del LinearLayout solo tenemos un elemento de tipo TextView. Este dispone de tres atributos. Los dos primeros definen el ancho y alto. En este caso el alto dependerá del texto que contenga el elemento TextView. El último indica el texto a mostrar. No se ha indicado un texto en concreto sino una referencia, "string/hello". Esta referencia ha de estar definida en el fichero res/values/strings.xml. Si abres este fichero, tienes el siguiente contenido:

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

    <string name="hello">Hello World, HolaCefireChesteActivity!</string>
    <string name="app_name">HolaCefireCheste</string>

</resources>
```

Esta es la práctica recomendada en Android para la inserción de textos dado que facilita su localización a la hora de realizar la traducción a otros idiomas. En resumen, los ficheros *layout* se utilizan para introducir el diseño de los interfaces y el fichero strings para introducir los textos utilizados en los distintos idiomas.

Si regresamos al fichero *HolaMundo.java* y deshacemos los cambios efectuados podemos analizar la línea que hemos comentado:

setContentView(R.layout.main);

Aquí, *R.layout.main* corresponde a un objeto *View* que será creado en tiempo de ejecución a partir del recurso *main.xml.* O sea,en *main.xml* definimos mediante código XML como queremos que sea la interfaz de la pantalla principal, y *R.layout.main* es la referencia a dicha interfaz. Trabajar de esta forma, en comparación con el diseño basado en código, no quita velocidad y requiere menos espacio. El *plug-in* de Eclipse crea automáticamente esta

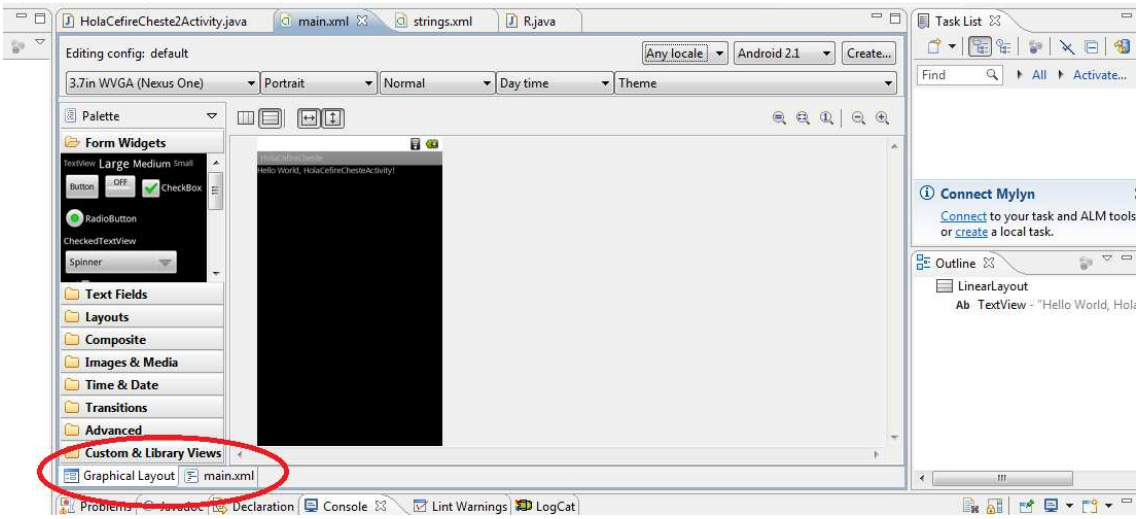
referencia en la clase *R* del proyecto a partir de todos los elementos de la carpeta *res.* El fichero *gen/R.java* contiene lo siguiente:

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.

package android.aguilar;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Este fichero se genera automáticamente. Nunca debe editarse de forma manual.

1.2.1.Edición visual de las vistas

Los *layouts* o ficheros de diseño en XML los podemos editar de forma visual. Si abrimos el fichero *res/layout/main.xml* podemos apreciar que en la parte inferior de la ventana central aparecen dos pestañas: Graphical Layout y main.xml. El *plug*-in de Android permite dos tipos de diseño: editar directamente el código XML (pestaña *main.xml)* o realizar el diseño de forma visual (pestaña *Graphical Layout)*.



En el marco derecho se visualiza una lista con todos los elementos de la vista. En este momento solo hay dos un *LinearLayout* que contiene un *TextView.* En el marco central aparece una representación de cómo se verá el resultado. En la parte superior aparecen varios controles para representar esta vista en diferentes configuraciones. Cuando diseñamos una

vista en Android, hay que tener en cuenta que desconocemos el dispositivo final donde será visualizada y por lo tanto debemos intentar que se visualice bien el cualquier configuración.

Para editar un elemento lo seleccionamos en el marco de la derecha o pincharemos sobre él en el marco central pudiendo de esta forma modificar algunas de sus propiedades.

El marco de la izquierda permite insertar de forma rápida nuevos elementos a la vista simplemente arrastrándolos.

1.2.2.Layouts

Si queremos combinar varios elementos de tipo vista tendremos que utilizar un objeto de tipo *Layout.* Un *Layout* es un contenedor de una o más vistas y controla su comportamiento y posición. Hay que destacar que un *Layout* puede contener a otro *Layout* y que es un descendiente de la clase *View.*

Los distintos *Layouts* con los que podemos trabajar son:

- **LinearLayout:** Es el Layout más utilizado en la práctica. Distribuye los elementos uno detrás de otro, bien de forma horizontal o vertical.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="CheckBox de prueba"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Botón de prueba"/>
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

</LinearLayout>
```

- **TableLayout:** Distribuye los elementos de forma tabular. Se utiliza la etiqueta *TableRow* cada vez que queremos insertar una nueva línea.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <TableRow >
        <AnalogClock
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <CheckBox
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="CheckBox de prueba" />
    </TableRow>

    <TableRow>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Botón de prueba" />
        <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/hello" />
    </TableRow>
</TableLayout>
```

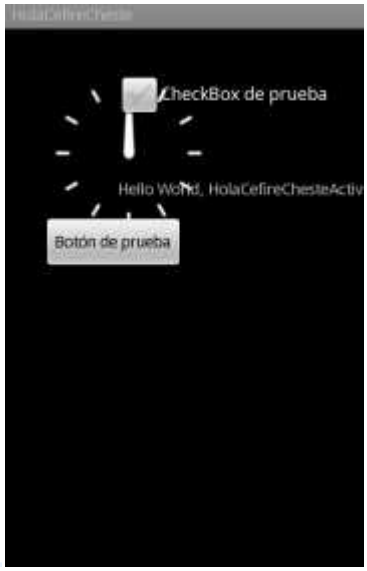
- **RelativeLayout:** Permite comenzar a situar los elementos en cualquiera de los cuatro lados del contenedor o ir añadiendo nuevos elementos pegados a estos.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <AnalogClock
        android:id="@+id/AnalogClock01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true" />
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="CheckBox de prueba"
        android:layout_below="@+id/AnalogClock01" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Botón de prueba"
        android:layout_alignParentRight="true" />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:layout_alignParentBottom="true"/>
</RelativeLayout>
```

Fijaos en este caso en las nuevas propiedades de los elementos que incorporamos en el Layout. Se trata de “layout_alignParentTop” para añadir el elemento en la parte superior del Layout, “layout_below” para añadir un elemento debajo de otro, “layout_alignParentRight” para añadir el elemento en la parte derecha del Layout, y por último en nuestro ejemplo, “layout_alignParentBotton” para añadir un elemento en la parte inferior del Layout.

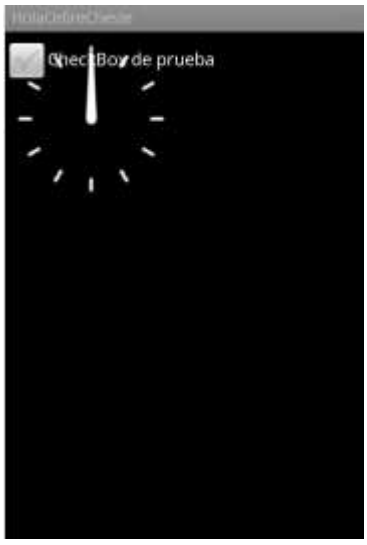
- **AbsoluteLayout:** Permite indicar las coordenadas (x,y) donde queremos que se visualice cada elemento.

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/AbsoluteLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="50px"
        android:layout_y="50px" />
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="CheckBox de prueba"
        android:layout_x="150px"
        android:layout_y="50px" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Botón de prueba"
        android:layout_x="50px"
        android:layout_y="250px" />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:layout_x="150px"
        android:layout_y="200px" />
</AbsoluteLayout>
```



- **FrameLayout:** Posiciona todos los elementos usando todo el contenedor, sin distribuirlos espacialmente. Este Layout suele utilizarse cuando queremos que varios elementos ocupen un mismo lugar pero solo uno será visible. Para modificar la visibilidad de un elemento utilizaremos la propiedad visibility:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/FrameLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="CheckBox de prueba" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Botón de prueba"
        android:visibility="invisible" />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:visibility="invisible" />
</FrameLayout>
```

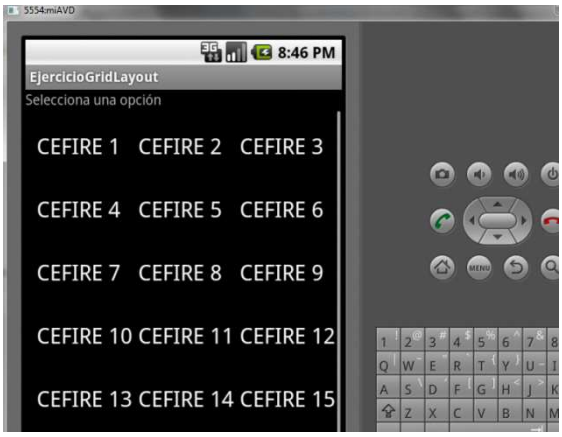


También podemos utilizar otras clases de Layouts:

- **ScrollView:** Visualiza una columna de elementos, cuando estos no caben en pantalla se permite un deslizamiento vertical mediante un scroll.
- **ListView:** Visualiza una lista deslizable verticalmente de varios elementos.
- **GridView:** Visualiza una cuadrícula deslizable de varias filas y varias columnas.
- **TabHost:** Proporciona una lista de ventanas seleccionables por medio de etiquetas que pueden ser pulsadas por el usuario para seleccionar la ventana que desea visualizar.
- **ViewFlipper:** Permite visualizar una lista de elementos de forma que se visualice uno cada vez. Puede ser utilizado para intercambiar los elementos cada cierto intervalo de tiempo.

1.2.2.1. Ejercicio Resuelto 1 (Código fuente en la carpeta EjercicioGridLayout)

Vamos a utilizar uno de estos últimos Layouts para conseguir una pantalla como la siguiente:



Los pasos son los siguientes:

1. **Incluir un layout GridView en /res/layout/main.xml con las propiedades deseadas para el mismo.**

```
<GridView
    android:id="@+id/gridView1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:numColumns="3" >
</GridView>
```

2. **Insertar los datos que queremos incluir en el layout en "EjercicioGridLayout.java" y dentro del método "onCreate". En nuestro caso se tratará de 25 cadenas de texto de la forma "CEFIRE X".**
    a. **Primero definimos un array genérico que contenga los datos de prueba.**
    b. **Después declaramos un adaptador de tipo ArrayAdapter pasándole un layout genérico.**
    c. **Por último asociamos el adaptador al control GridView mediante su método setAdapter.**

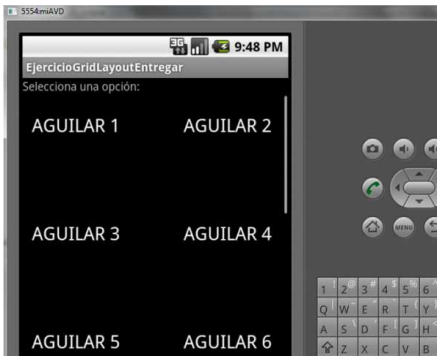
```
//Insertamos los datos en el layout GridView
//Primero definimos array genérico que contenga nuestros datos de prueba
String[] datos = new String[25];
for(int i=1; i<=25; i++)
    datos[i-1] = "CEFIRE " + i;
//Declaramos un adaptador de tipo ArrayAdapter pasándole un layout genérico
ArrayAdapter<String> adaptador =
        new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, datos);
//Asociamos el adaptador al control GridView mediante su método setAdapter.
GridView grdOpciones = (GridView)findViewById(R.id.gridView1);
grdOpciones.setAdapter(adaptador);
```

1.2.2.2. Ejercicio 1

Sabiendo que las propiedades más importantes de los GridLayout son las siguientes:

Propiedad	Descripción
android:numColumns	Indica el número de columnas de la tabla o "auto_fit" si queremos que sea calculado por el propio sistema operativo a partir de las siguientes propiedades.
android:columnWidth	Indica el ancho de las columnas de la tabla.
android:horizontalSpacing	Indica el espacio horizontal entre celdas.
android:verticalSpacing	Indica el espacio vertical entre celdas.
android:stretchMode	Indicar qué hacer con el espacio horizontal sobrante. Si se establece el valor columnWidth este espacio será absorbido a partes iguales por las columnas de la tabla. Si por el contrario se establece a spacingWidth será absorbido a partes iguales por los espacios entre celdas.

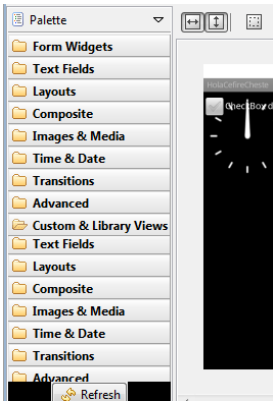
Modificar el ejercicio resuelto 1 para que la pantalla nos quede de la siguiente forma.



Debéis modificar la cadena "AGUILAR" por vuestro primer apellido y enviar una captura de pantalla.

1.2.3.Controles gráficos

Hemos estudiado los Layouts que nos permiten distribuir elementos y controles de la interfaz por la pantalla del dispositivo. Disponemos de una cantidad de controles enorme para poder insertar en nuestras pantallas. En la vista gráfica de Eclipse podemos encontrar una clasificación de los distintos tipos:



Veremos a continuación algunos de estos controles.

1.2.3.1. Botones

Disponemos de tres tipos de botones: el clásico (Button), el de tipo on/off (ToggleButton), y el que puede contener una imagen (ImageButton).

Button
Es el tipo de botón más básico.
	`<Button`
	`android:id="@+id/button1"`
	`android:layout_width="wrap_content"`
	`android:layout_height="wrap_content"`
	`android:text="Cefire Cheste"`
	`android:textColor="#ff00fe"`
	`android:textSize="40"`
	`android:textStyle="bold" />`
		

ToggleButton
Puede permanecer en dos estados: pulsado/no pulsado
	`<ToggleButton`
	`android:id="@+id/toggleButton1"`
	`android:layout_width="wrap_content"`
	`android:layout_height="wrap_content"`
	`android:textOn="PULSADO"`
	`android:textOff="NO PULSADO"`
	`android:textSize="40"`
	`android:checked="true" />`
		

ImageButton
Podemos definir una imagen a mostrar en lugar de un texto. La imagen debe ser un recurso incluido en la carpeta /res/drawable.
	`<ImageButton`
	`android:id="@+id/imageButton1"`
	`android:layout_width="wrap_content"`
	`android:layout_height="wrap_content"`
	`android:src="@drawable/ok" />`
		

- **EVENTOS DE UN BOTÓN**

El más común de todos los eventos es el evento onClick. Para definir la lógica de este evento tendremos que implementarla definiendo un nuevo objeto View.onClickListener() y asociándolo al botón mediante el método setOnClickListener():

```
final Button btnBoton1 = (Button)findViewById(R.id.BtnBoton1);
final TextView lblMensaje = (TextView)findViewById(R.id.LblMensaje);
btnBoton1.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View arg0)
{
lblMensaje.setText("Botón 1 pulsado!");
}
});
```

En el caso de un ToggleButton nos será de utilidad saber en qué estado queda el botón después de ser pulsado. Para ellos podemos utilizar el método isChecked().

```
final ToggleButton btnBoton2 = (ToggleButton)findViewById(R.id.toggleButton1);
final TextView lblMensaje = (TextView)findViewById(R.id.LblMensaje);
btnBoton2.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View arg0)
{
    if (btnBoton2.isChecked())
        lblMensaje.setText("Botón 2 ESTADO ON!");
    else
        lblMensaje.setText("Botón 2 ESTADO OFF!");
}
```

1.2.3.2. Imágenes, etiquetas y cuadros de texto

En la siguiente tabla vemos cómo utilizar diferentes controles tanto en la interfaz como en la lógica de la aplicación. Debemos elegir una de las dos formas. Si elegimos en la interfaz, incluiremos el código en un fichero xml dentro de res/layout. Si elegimos en la lógica de la aplicación incluiremos el código en un fichero java.

CONTROL	EJEMPLO DE USO	PROPIEDADES INTERESANTES

ImageView
Permite mostrar imágenes en la aplicación.
En la propiedad *src* definimos la imagen a mostrar.
	En la interfaz:
	`<ImageView android:id="@+id/ImgFoto"`
	`android:layout_width="wrap_content"`
	`android:layout_height="wrap_content"`
	`android:src="@drawable/icon"`
	`android:layout_below="@id/BtnNegrita"`
	`android:layout_alignParentRight="true" />`
	En la lógica de la aplicación:
	`ImageView img = (ImageView)findViewById(R.id.ImgFoto);`
	`img.setImageResource(R.drawable.icon);`
		android:src
		android:maxWidth
		android:maxHeight

TextView
Son las etiquetas de texto. Se utiliza para mostrar un determinado texto al usuario.
	En la interfaz:
	`<TextView android:id="@+id/LblEtiqueta"`
	`android:layout_width="fill_parent"`
	`android:layout_height="wrap_content"`
	`android:text="Introduce Cefire al que perteneces:"`
	`android:background="#7733AA"`
	`android:typeface="monospace"/>`
	En la lógica de la aplicación:
	`//Recuperamos el texto de una etiqueta`
	`final TextView lblEtiqueta = (TextView)findViewById(R.id.LblEtiqueta);`
	`//Le concatenamos unos números`
	`String texto = lblEtiqueta.getText().toString();`
	`texto += "000";`
	`//Actualizamos su contenido`
	`lblEtiqueta.setText(texto);`
	`//Cambiamos su color de fondo`
	`lblEtiqueta.setBackgroundColor(0);`
		android:text
		android:background
		android:textColor
		android:textSize
		android:typeFace

EditText
Permite la introducción y edición de texto por parte del usuario .
	En la interfaz:
	`<EditText android:id="@+id/TxtTexto"`
	`android:layout_width="fill_parent"`
	`android:layout_height="wrap_content"`
	`android:layout_below="@id/LblEtiqueta"`
	`android:text="Escribe aquí tu nombre" />`
	En la lógica de la aplicación:
	`final EditText txtTexto = (EditText)findViewById(R.id.TxtTexto);`
	`//Recuperamos el texto mediante getText()`
	`String texto = txtTexto.getText().toString();`
	`//Establecemos otro texto mediante setText(nuevaCadena)`
	`txtTexto.setText("Hola alumnos del Cefire");`
		android:text
		android:textColor
		android:textSize
		android:height
		android:width

1.2.3.3. Texto con formato en controles TextView y EditText

Tenemos la posibilidad de cargar un cuadro de texto (EditText) o una etiqueta (TextView) a partir de un fragmento de texto en formato HTML. Para ello podemos utilizar el método Html.fromHtml(String) de la siguiente forma:

```
//Asigna texto con formato HTML
txtTexto.setText(
Html.fromHtml("<p>Esto es un <b>simulacro</b>.</p>"),
BufferType.SPANNABLE);
```

También podemos realizar la operación contraria mediante el método Html.toHtml(Spannable) de la siguiente forma:

```
//Obtiene el texto del control con etiquetas de formato HTML
String aux2 = Html.toHtml(txtTexto.getText());
```

1.2.3.4. Checkbox y RadioButton

Control	Ejemplo de uso	Propiedades interesantes

CheckBox
Se suele utilizar para marcar y desmarcar opciones en una aplicación.
	En la interfaz:
	`<CheckBox android:id="@+id/ChkMarcame"`
	`android:layout_width="wrap_content"`
	`android:layout_height="wrap_content"`
	`android:text="Márcame!" />`
	En la lógica de la aplicación:
	`if (checkBox.isChecked()) {`
	`checkBox.setChecked(false);`
	`}`
	Eventos que puede lanzar este control:
	`final CheckBox cb = (CheckBox)findViewById(R.id.chkMarcame);`
	`cb.setOnCheckedChangeListener(`
	`new CheckBox.OnCheckedChangeListener() {`
	`public void onCheckedChanged(CompoundButton buttonView,`
	`boolean isChecked) {`
	`if (isChecked) {`
	`cb.setText("Checkbox marcado!");`
	`}`
	`else {`
	`cb.setText("Checkbox desmarcado!");`
	`}`
	`}`
	`});`
		Las mismas que para el control TextView.

<table>
<tr>
<td>RadioButton
Dentro del grupo de opciones solo tendremos una marcada.</td>
<td>En la interfaz:

```
<RadioGroup android:id="@+id/gruporb"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <RadioButton android:id="@+id/radio1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 1" />

    <RadioButton android:id="@+id/radio2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 2" />

</RadioGroup>
```

En la lógica de la aplicación:

```
final TextView lblMensaje = (TextView)findViewById(R.id.lblSeleccion);

final RadioGroup rg = (RadioGroup)findViewById(R.id.gruporb);
//Desmarcamos todas las opciones
rg.clearCheck();
//Marcamos una opción determinada mediante su ID
rg.check(R.id.radio1);
//Obtenemos el ID de la opción marcada
int idSeleccionado = rg.getCheckedRadioButtonId();
```

Eventos que puede lanzar este control:

```
rg.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
    //Informamos de los cambios en el elemento seleccionado
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        lblMensaje.setText("ID opción seleccionada: " + checkedId);
    }
});
```

</td>
<td>Las mismas que para el control TextView.</td>
</tr>
</table>

1.2.3.5. Listas desplegables

1.2.3.5.1. Adaptadores en Android (adapters)

Los adaptadores son una interfaz común al modelo de datos que existe por detrás de varios controles.

En nuestro caso, las listas accederán a los datos que contienen a través de un adaptador.

El más sencillo de todos los adaptadores es el ArrayAdapter que provee de datos a un control de selección (o sea, a una lista) a partir de un array de objetos de cualquier tipo.

La creación de un adaptador sería la siguiente:

```
//Definición del array Java que contendrá los datos a mostrar en el control
final String[] datos =
    new String[]{"Elem1","Elem2","Elem3","Elem4","Elem5"};
//Creación del adaptador al que pasamos 3 parámetros
//1. El contexto, que normalmente será una referencia a la actividad
//   donde se crea el adaptador.
//2. El ID del layout sobre el que se mostrarán los datos del control.
//   En este caso le pasamos el ID de un layout predefinido en Android
//   (android.R.layout.simple_spinner_item), formado únicamente por un
//   control TextView, pero podríamos pasarle el ID de cualquier layout
//   de nuestro proyecto con cualquier estructura y conjunto de controles.
//3. El array que contiene los datos a mostrar.
ArrayAdapter<String> adaptador =
    new ArrayAdapter<String>(this,
        android.R.layout.simple_spinner_item, datos);
```

Después solo tenemos que asignar este adaptador a nuestro control de selección para que éste muestre los datos en la aplicación.

1.2.3.5.2. Control Spinner

El control Spinner es una lista desplegable que muestra una especie de lista emergente al usuario con todas las opciones disponibles y al seleccionarse una de ellas ésta queda fijada en el control.

El código que debemos añadir en la interfaz es el siguiente:

```
<Spinner android:id="@+id/CmbOpciones"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

Las opciones para personalizar el aspecto visual del control (fondo, color y tamaño de fuente, etc.) son las mismas ya comentadas para los demás controles.

El código que debemos utilizar para enlazar el adaptador (o sea, los datos) con el control es el siguiente:

```
//Obtenemos una referencia al control a través de su ID
final Spinner cmbOpciones = (Spinner)findViewById(R.id.CmbOpciones);
//Podemos personalizar también el aspecto de cada elemento en la lista emergente
adaptador.setDropDownViewResource(
        android.R.layout.simple_spinner_dropdown_item);
//Asignamos el adaptador al control
cmbOpciones.setAdapter(adaptador);
```

El evento lanzado por el control Spinner más utilizado es el que se genera al seleccionarse una opción de la lista desplegable (onItemSelected). Un ejemplo de uso sería el siguiente:

```
cmbOpciones.setOnItemSelectedListener(
    new AdapterView.OnItemSelectedListener() {
        //Este método será llamado cada vez que se seleccione una opción
        //en la lista desplegable
        public void onItemSelected(AdapterView<?> parent,
            android.view.View v, int position, long id) {
                lblMensaje.setText("Seleccionado: " + datos[position]);
        }
        //Este método será llamado cuando no haya ninguna opción seleccionada.
        //Esto puede ocurrir por ejemplo si el adaptador no tiene datos.
        public void onNothingSelected(AdapterView<?> parent) {
            lblMensaje.setText("");
        }
});
```

La lista quedaría de la siguiente forma:

2. APLICACIÓN DE EJEMPLO: SOLOBICI DIDÁCTICO

A lo largo del curso vamos a ir construyendo una aplicación de ejemplo con los conocimientos que vayamos adquiriendo.

Puesto que me gusta mucho montar en bicicleta y puedo observar que el respeto que existe hacia este tipo de vehículo en la carretera es prácticamente nulo, la aplicación que vamos a desarrollar se trata de un juego en el que manejaremos una bicicleta y tenemos que ir "destruyendo" los coches y las motos que nos encontremos. Lo haremos lanzando ruedas de bicicleta a modo de disparo.

Os habréis dado cuenta que el título de la aplicación se compone también de la palabra "didáctico". Esto no es porque destruir coches y motos sea didáctico sino porque incorporaremos una parte al juego en la que aparecerán una serie de preguntas sobre algún tema que nos interese que nuestros alumnos respondan y según el número de preguntas acertadas tendrán más o menos vidas para jugar. Yo utilizo este juego para que mis alumnos se jueguen una partidita al finalizar cada uno de los temas que vemos en clase, y así de paso repasan mediante las preguntas muchas cuestiones de las vistas en ese tema.

Las preguntas podrán incluso recuperarse de un servidor de Internet que habilitemos para esta tarea, donde almacenaremos un repositorio de preguntas.

En este tema crearemos una serie de vistas que nos permitirán diseñar una sencilla interfaz de usuario.

Los datos del proyecto nuevo que debemos generar serán los siguientes:

Project name: Solobici
Build Target: Android 2.1
Application name: Aplicacion
Package name: juego.solobici
Create Activity: Solobici
Min SDK Version: 7

2.1. Ejercicios

2.1.1. Ejercicio Resuelto 2

Crear una vista similar a la que tenemos a continuación. Está formada por un LinearLayout que contiene un TextView y cuatro Button. Trata de utilizar recursos (Fichero res/values/strings.xml) para introducir los cinco textos que aparecen.

SOLUCIÓN:

El código XML (Archivo res/layout/main.xml) debe ser similar al siguiente:

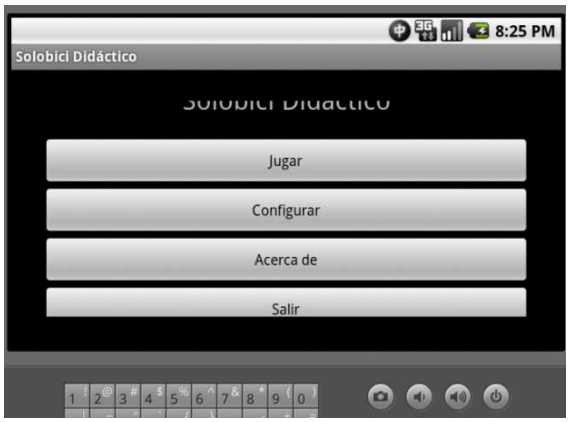
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="30dip" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/tituloAplicacion"
        android:gravity="center"
        android:textSize="25sp"
        android:layout_marginBottom="20dip" />

    <Button
        android:id="@+id/Boton01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/Jugar" />


    <Button
        android:id="@+id/Boton02"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/Configurar" />

    <Button
        android:id="@+id/Boton03"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/Acercade" />

    <Button
        android:id="@+id/Boton04"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/Salir" />

</LinearLayout>
```

El fichero res/values/strings.xml debe ser similar al siguiente:

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

    <string name="hello">Hello World, Solobici!</string>
    <string name="app_name">Solobici Didáctico</string>
    <string name="tituloAplicacion">Solobici Didáctico</string>
    <string name="Jugar">Jugar</string>
    <string name="Configurar">Configurar</string>
    <string name="Acercade">Acerca de</string>
    <string name="Salir">Salir</string>

</resources>
```

2.1.2.Ejercicio 2

Los teléfonos móviles basados en Android permiten cambiar la configuración en apaisado y en vertical. Para conseguir este efecto con el emulador pulsa Ctrl+F11. Si te das cuenta, el resultado de la vista que acabamos de realizar en vertical, no queda todo lo bien que nos gustaría en horizontal.



Android permite diseñar una vista diferente para la configuración horizontal y vertical.

Pasos a seguir:

1. Crea la carpeta res/layout-land.
2. Copia en ella el fichero main.xml
3. Crea una vista similar a la que ves a continuación: está formada por un LinearLayout que contiene un TextView y un TableLayout con dos Button por fila.
   Los atributos que puedes utilizar para la etiqueta TableLayout pueden ser los siguientes:

```
<TableLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center"
    android:stretchColumns="*">
```

   Podéis consultar la documentación de la clase TableLayout en http://developer.android.com/reference/android/widget/TableLayout.html.

Nos debe quedar algo como esto:

2.1.3.Ejercicio Resuelto 3

Implementar la caja *Acerca de*.

Crearemos la caja Acerca de… y haremos que aparezca al pulsar el botón correspondiente. Los pasos a seguir son los siguientes:

1. Añadir la siguiente cadena al fichero res/values/strings.xml

```
<string name="TextoAcercaDe">Esta aplicación ha sido desarrollada para el curso del Cefire
    de Cheste sobre programación de dispositivos móviles mediante Android.</string>
```

2. Crear el fichero res/layout/acercade.xml. Botón derecho sobre la carpeta res/layout, pulsar en *New->Other*. Seleccionar *Android->Android XML File* y escribir en File *acercade.xml*. Este fichero debe contener el siguiente código:

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

<TextView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/TextoAcercaDe" />
```

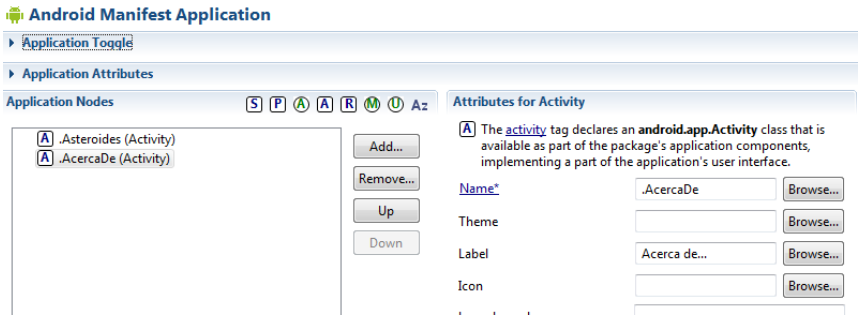
3. Crear una nueva actividad para visualizar esta vista. Creamos por tanto la clase AcercaDe.java en el paquete *src/juego.solobici* que debe contener el siguiente código:

```
package juego.solobici;

import android.app.Activity;
import android.os.Bundle;

public class AcercaDe extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        //Hacemos visible la interfaz/layoutque se encuentra en acercade.xml
        setContentView(R.layout.acercade);
    }
}
```

4. Creamos un controlador de botón en la clase Solobici.java para que se lance la nueva actividad al pulsar sobre dicho botón.

```
package juego.solobici;

import android.app.Activity;

public class Solobici extends Activity {

    private Button bAcercaDe;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //Botón y escuchador para la pantalla "Acerca de"
        //Al hacer click en este botón llamamos al método lanzarAcercaDe()
        bAcercaDe = (Button) findViewById(R.id.Boton03);
        bAcercaDe.setOnClickListener(new OnClickListener(){
            public void onClick(View view) {
                lanzarAcercaDe();
            }
        });
    }

    //Método que activa la pantalla AcercaDe
    public void lanzarAcercaDe(){
        Intent i = new Intent(this, AcercaDe.class);
        startActivity(i);
    }
}
```

5. Cualquier actividad nueva debe registrarse en AndroidManifest.xml. Para hacer esto abrimos el archivo AndroidManifest.xml y seleccionamos la pestaña Application. En Application Nodes pulsar el botón Add… y seleccionamos Activity. Rellenar los campos de la derecha de la siguiente forma:



6. A continuación le vamos a aplicar un tema al mensaje mostrado para que visualmente sea un poco más atractivo ya que hasta este momento si ejecutáis la aplicación y pulsáis el botón de “Acerca de” podréis comprobar que se abre una nueva pantalla con el mensaje que hemos decidido pero visualmente no es muy llamativa.
   Un tema no es más que un conjunto de estilos (parecido a CSS) que cambian el aspecto de una de nuestras vistas. En este caso vamos a utilizar un tema de Android. Más adelante veremos cómo crear nuestros propios temas.
   Abrimos *AndroidManifest.xml* y seleccionamos la pestaña *Application*. En *Application Nodes* pulsamos sobre .AcercaDe e introducimos en el campo *Theme* el valor: @android:style/Theme.Dialog.
   También podríamos hacerlo directamente en el fichero *AndroidManifest.xml* de forma que incluyese lo siguiente:

```
<activity android:name=".AcercaDe"
    android:label="Acerca de..."
    android:theme="@android:style/Theme.Dialog">
</activity>
```

Obtenemos lo siguiente:

Bueno, pues de momento ya tenemos una interfaz básica en nuestra aplicación/juego. En los siguientes temas iremos introduciendo nuevas cosas para intentar completarla.