Desarrollo para dispositivos móviles con Android: programación de aplicaciones básicas.

Contenido

1 Introducción

Las aplicaciones Android dependen en gran medida del archivo de características *AndroidManifest.xml* (así como otros archivos XML asociados) para especificar la configuración, lo que no se corresponde directamente con la programación de la aplicación en sí.

En una aplicación Android, si se sigue una analogía con una aplicación de escritorio, cada ventana es una actividad (*activity*). Las actividades se describen en el archivo de características, mientras que el código correspondiente se crea en una clase derivada de **Activity**. Por otra parte, el código de la interfaz de usuario de la actividad es un archivo XML asociado a ella, con su mismo nombre, y residiendo en *res/layouts/*. Cada componente de la interfaz de una actividad de Android se describe como etiquetas dentro de este archivo.

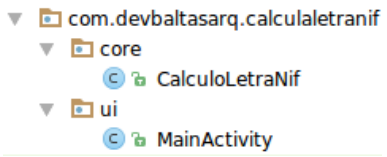
En este tema, se repasará la programación básica en android creando una pequeña aplicación que calcula la letra correspondiente a un determinado DNI.

2 Lógica de negocio y diseño

Es muy interesante separar la lógica de negocio de la interfaz de usuario y su manejo. Aunque el código necesario para calcular la letra del NIF es muy sencillo, siempre se obtiene un mejor diseño factorizando los fuentes, de manera que no existan dependencias directas entre la vista y el comportamiento.

La clase que realiza el cálculo de la letra del NIF correspondiente a un DNI se muestra a continuación. Se basa en obtener una letra de un alfabeto determinado a partir del número de DNI módulo 23.

```
package com.devbaltasarq.calculaletranif.core;

public class CalculoLetraNif {

        public static final String NIF_STRING_ASOCIATION = "TRWAGMYFPDXBNJZSQVHLCKE";

        /**
         * Devuelve la letra correspondiente al NIF
         * @param dni dni del que calcular la letra del NIF, como un int
         * @return La letra del nif, como un char.
         */
        public static char calculaLetraNif(int dni) {
          return NIF_STRING_ASOCIATION.charAt( dni % 23 );
        }
}
```



Es una buena idea separar el código que se corresponde con la lógica de negocio, del código que se corresponde con la interfaz de usuario. Así crearemos dos *packages* nuevos colgando de **com.xxx.calculaletranif**: **core** y **ui**. Dentro del primero, colocaremos la clase **CalculoLetraNif**. Moveremos también **MainActivity** al package **ui**, modificando de forma correspondiente la primera línea (debe quedar como package *com.devbaltasarq.calculaletranif.ui;*).

3 Creando una actividad

Si bien los distintos entornos de programación automatizan ya la creación de tareas, no está de más conocer cómo se crean actividades de forma manual. El punto de partida es el archivo *AndroidManifest.xml*, donde se registran todas las actividades.

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

```
    package="com.devbaltasarq.calculaletranif"
    android:versionCode="1"
    android:versionName="1.0" >
...
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Nótese el atributo *package* dentro del primer elemento, **Manifest**. Es este atributo el que le dice al compilador dónde se encuentran las clases relevantes para la aplicación (por ejemplo, el *core* no le importa). Según la modificación que hicimos anteriormente, debemos cambiar el contenido del atributo package de "*com.devbaltasarq.calculaletranif*", a "*com.devbaltasarq.calculaletranif.ui*".

Los *intents* describen qué acciones puede soportar una actividad. En este caso, la actividad es la principal de la aplicación, y al tener además la categoría *launcher*, indica que aparecerá en el menú principal de aplicaciones del dispositivo.

La etiqueta *android:name* indica el nombre de la actividad (siempre empieza por punto para concatenarla con el atributo package comentando más arriba), y *android:label* el título que aparecerá en la barra superior.

Además de la descripción de la actividad, será necesario crear código ejecutable para ella. Para ello, es necesario escribir la clase **MainActivity** (debe coincidir con *android:name*) derivando de **Activity** o de **AppCompatActivity** (esta es la que permite que una aplicación tenga la misma apariencia sin importar la versión del dispositivo), y una layout del mismo nombre, pero con extensión xml (que lleve el mismo nombre no es estrictamente obligatorio, pero ayuda a identificar correctamente cada componente). Google sugiere comenzar los nombres por "activity_". Por ejemplo, el *layout* de **MainActivity** debería ser *activity_main.xml*.

```
public class MainActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate( savedInstanceState );
        setContentView( R.layout.activity_main );
    }
}
```

El método *onCreate()* es el primero en ser ejecutado cuando la actividad arranca. Lo primero es mantener el comportamiento por defecto llamando a *super.onCreate()*.

En esta explicación han aparecido ya ciertas constantes (como *R.layout…*, o *@strings…*) que se explican en detalle en la siguiente sección: someramente, identifican componentes de la interfaz de usuario.

Finalmente, cada actividad lleva asociado un archivo XML en el que se describe su interfaz. El comienzo del archivo no identifica ser un layout de una actividad, simplemente arranca con el componente más externo, que Android Studio pone como siempre **RelativeLayout**.

En realidad, no es necesario conocer a fondo los detalles de este archivo XML, ya que el diseñador integrado hace la mayor parte del trabajo por nosotros. En cualquier caso, siempre será interesante saber qué está ocurriendo desde el diseñador, de forma que se pueda ir al texto y arreglarlo.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

       <!-- Más cosas... -->

</RelativeLayout>
```

Por ejemplo, podríamos crear desde el archivo XML una muy simple interfaz que contenga una etiqueta, una entrada de texto, dos etiquetas, y un botón. Retomamos el hueco indicado en el código XML anterior como “más cosas...”. El contenedor más simple a utilizar es el contenedor horizontal, que agrupará los elementos en una sola línea.

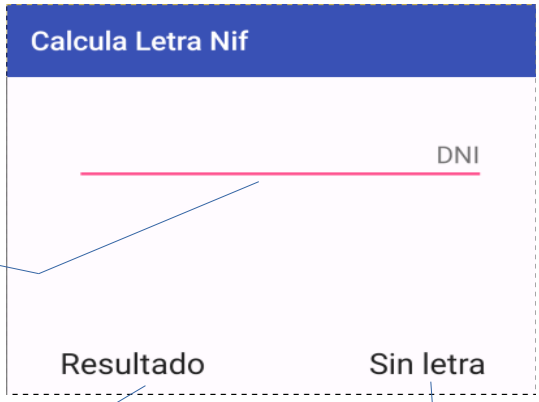
```
       <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <EditText
            android:id="@+id/edDni"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:ems="8"
            android:inputType="number"
            android:hint="DNI"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="NIF:" />

        <TextView
            android:id="@+id/lblResult"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Sin letra"/>
    </LinearLayout>
```

En este sencillo ejemplo se crean tres elementos: un **EditText** o campo de entrada de texto (que solo permitirá introducir números), una etiqueta o **TextView** que visualizará el mensaje fijo “NIF:”, y finalmente, otra etiqueta que mostrará el resultado final.

Con el diseñador se pueden hacer muchas más cosas, claro, aunque el germen de la interfaz de usuario sea el mismo. El ejemplo completo puede verse a continuación. Recuerda que el

RelativeLayout lo pone el propio entorno. En negrita, están marcados los componentes y los atributos más importantes.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="20dp">

        <LinearLayout android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_weight="0.33"
            android:orientation="horizontal">

            <EditText android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_marginLeft="10dp"
                android:id="@+id/edDni"
                android:inputType="number"
                android:gravity="right"
                android:hint="@string/label_value"/>
        </LinearLayout>
        <LinearLayout android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_weight="0.33"
            android:orientation="horizontal">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/label_result"
                android:textAppearance="@android:style/TextAppearance.Large"/>
            <TextView
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_marginLeft="10dp"
                android:text="@string/label_default_result"
                android:id="@+id/lblResult"
                android:textAppearance="@android:style/TextAppearance.Large"
                android:gravity="right"/>
        </LinearLayout>

        <LinearLayout android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_weight="0.34"
            android:orientation="horizontal">
        </LinearLayout>

    </LinearLayout>
</RelativeLayout>
```

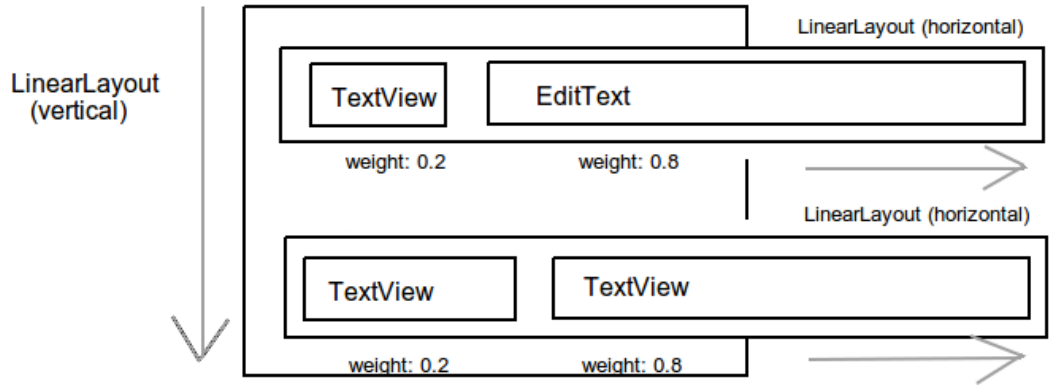


Para comprender por qué aparecen entre comillas mensajes como “@string/label_value”, en lugar de un texto directo, es necesario referirse a la siguiente sección sobre recursos.

El contenido de cada etiqueta ya no es necesario escribirlo (aunque se puede hacerlo), solamente es necesario utilizar el diseñador asociado. En cuanto a los layouts y los componentes gráficos, se introducirán un poco más abajo.

3.1 Los layouts

Los *layouts* permiten que los componentes adopten una disposición ordenada de alguna manera. Si no existieran los *layouts*, cada componente aparecería uno detrás de otro, provocando confusión y desorden. Conocemos ya **RelativeLayout**, que permite colocar componentes con respecto a otros que ya existen, y **LinearLayout**. Este último es el que exploraremos en profundidad, teniendo en cuenta que tiene dos versiones: la horizontal y la vertical. En realidad, a base de anidar **LinearLayout**'s, es posible crear interfaces ya bastante avanzadas.



En la imagen superior se aprecia como dos **LinearLayout** horizontales están anidados en un **LinearLayout** vertical, reproduciendo la interfaz de usuario de la sección anterior. La forma de indicar cuánto espacio ocupa cada componente es especificar un número entre 0 y 1 en su atributo *weight*. Para indicar cómo se distribuyen los componentes en el interior de un componente contenedor, se modifica el atributo *gravity*. Por ejemplo, las etiquetas de texto de la segunda línea (el segundo **layout** horizontal), tienen gravity *left* y *right*, respectivamente.

4 Los recursos

Los recursos se sitúan dentro de la subcarpeta *res/*. Cuando se explicaba el archivo *AndroidManifest.xml*, en lugar de aparecer el título de la actividad directamente en *android:title*, aparece el texto: *@string/title_activity_main*. El objetivo es tener todo el texto reunido en una tabla, de manera que posibles traducciones de la aplicación resulten más sencillas. Esta tabla forma parte de los recursos de la aplicación, en la que se encuadran también, por ejemplo, los iconos de la aplicación en diferentes tamaños. A continuación, se puede apreciar el contenido del archivo *strings.xml*.

```
<resources>
    <string name="app_name">com.devbaltasarq.calculaletranif</string>
    <string name="title_activity_main">Letra del Nif</string>
    <string name="dni_placeholder">DNI</string>
    <string name="label_enter_dni">Introduzca Dni</string>
    <string name="label_result">Resultado:</string>
    <string name="label_default_result">Sin letra</string>
</resources>
```

También es interesante indicar que el archivo XML de descripción de la interfaz, asociado a la actividad, visto más arriba, se guarda en *res/layout*.

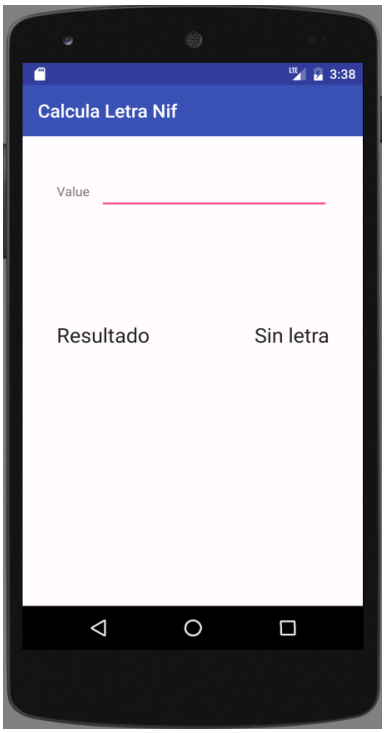
En las subcarpetas *res/drawable-xxx*, se guardan los iconos de la aplicación para diferentes resoluciones. Por ejemplo, en la carpeta *res/drawable-ldpi* se guarda el icono para pantallas con bajas resoluciones.

5 Constantes

Además del código fuente y los recursos que debemos proveer al SDK para que sea capaz de crear una aplicación para Android, el propio SDK genera unas constantes que hacen que sea posible programar relacionando todos los componentes. De ahí que, en algunas ocasiones, sea necesario recompilar para que se resuelvan ciertos errores indicando que cierto recurso no existe. Esta información sobre la aplicación, que se puede utilizar desde el código fuente, está accesible desde la constante **R**.

Por ejemplo, podemos observar en el código fuente que en el constructor de la clase **Main**, se asigna su propia interfaz para la visualización mediante la instrucción: *setContentView(R.layout.activity_main);*. Efectivamente, todas las actividades son identificables, accesibles, desde *R.layout_activity*. Y los componentes gráficos dentro de cada actividad, son accesibles desde *R.id*. Por ejemplo, si creamos un componente gráfico (una vista (*view)*, según Android), con el nombre o *id*: *edDNI*, entonces podremos referenciarlo desde el código fuente como *R.id.edDNI*.

6 Creando una interfaz para la actividad



Es posible crear la interfaz modificando directamente el archivo XML de *layout* correspondiente a la misma, o emplear el diseñador integrado en el propio entorno. El componente gráfico más básico es, probablemente, **TextView**, que permite visualizar texto en la pantalla, y las formas de entrada también más básicas son **EditText** (permite introducir texto de diferentes tipos), y **Button** (muestra un botón que se puede pulsar). Y de hecho, con estos componentes tan básicos tenemos más que de sobra para crear la aplicación que calcula la letra del NIF. De hecho, la interfaz de la aplicación es tan simple como la que se puede ver en la figura.

6.1 Layouts

La forma de agrupar varios componentes gráficos es la de incluirlos dentro de *layouts*. Como ya hemos visto, el más sencillo de utilizar es **LinearLayout** en sus dos versiones: horizontal y vertical. Dado que se pueden anidar todo lo que se desee, se puede construir una especie de tabla con un número de columnas por cada fila totalmente libre: independientemente de la orientación horizontal o vertical, esta disposición o *layout* organiza los componentes unos detrás de otros.

Así, la disposición de la interfaz será la siguiente:

- LinearLayout (vertical)
  - LinearLayout (horizontal)
    - lblValue (TextView: “Introduzca DNI”)
    - edValue (EditText)
  - LinearLayout (horizontal)
    - lblValue (TextView: “Resultado:”)
    - lblValue (TextView: “Sin letra”)

6.2 Unidades

Durante la creación de la interfaz, será necesario referirse a distancias o tamaños. Las unidades en las que se puede proporcionar esta información varía, como se puede ver en esta tabla.

Densidad	Descripción	Unidades por pulgada real	Independiente de la densidad	Mismo tamaño real en cualquier pantalla
px	Píxeles	Varía	No	No
in	Pulgadas	1	Sí	Sí
mm	Milímetros	24.5	Sí	Sí
pt	Puntos	72	Sí	Sí
dp	Píxeles independientes de la densidad	~ 160	Sí	No
sp	Píxeles independientes de la escala	~ 160	Sí	No

Descripción detallada:

- **px**: Píxeles – se corresponden con los píxeles en la pantalla real.
- **in**: Pulgadas(*inches*) – basadas en el tamaño de la pantalla real. 1 pulgada = 2.54 centimetros
- **mm**: Millímetros – basados en el tamaño de la pantalla real.
- **pt**: Puntos – 1/72 de una pulgada según el tamaño de la pantalla real.
- **dp**: Píxeles independientes de la densidad (*Density-independent Pixels*) – una unidad abstracta basada en el tamaño real de la pantalla. Estas unidades son relativas a una pantalla de 160 dpi (puntos por pulgada), así que un *dp* es un píxel en una pantalla de 160dpi. La relación de *dp*'s a píxeles cambiará con la densidad de la pantalla, pero no necesariamente en una proporción directa.
- **sp**: Pixeles independiente de la escala – Es una unidad de medida similar al *dp*, pero además escalada según las preferencias de fuentes del usuario. Así, depende tanto de la densidad como de las preferencias del usuario.

En general, lo aconsejable es utilizar medidas en sp's para las fuentes, y en dp's para todo lo demás. De esta forma, la aplicación será comptabile con todos los tamaños de pantalla.

6.3 Escribiendo la interfaz

Hemos visto el archivo XML correspondiente a la interfaz en secciones anteriores. Las modificaciones que se realicen en este XML repercuten directamente en el diseñador, y viceversa.

En negrita se marcan los comienzos de las etiquetas principales: tres *layouts*, uno vertical y dos horizontales; un **TextView**, y un **EditText**, y finalmente dos **TextView** que permiten mostrar el resultado (mientras uno siempre se mantendrá fijo en pantalla, el otro mostrará la letra resultante del cálculo). Los elementos que van a manejarse desde el programa necesitan un identificador (*android:id*), mientras que el resto, dado que van a ser estáticos, no. Así, el identificador para el penúltimo **TextView** no es necesario, aunque no penaliza en absoluto indicarlo. Existen dos componentes claves aquí:

```
<EditText android:id="@+id/edDni"/>
<TextView android:id="@+id/lblLetra" android:text="@string/label_default_result" />
```

Del componente *edDni* obtendremos el número del DNI para el cálculo, mientras que en el componente *lblLetra* visualizaremos el resultado. Es interesante darles algunas propiedades más a los cuatro componentes activos:

```
<TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/label_enter_dni"
            android:layout_gravity="center_vertical"/>
<EditText
            android:id="@+id/edDni"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:ems="8"
            android:inputType="number" android:layout_gravity="center_vertical" android:gravity="right"
            android:hint="@string/dni_placeholder"/>
```

Resulta muy interesante la propiedad *android:inputType* para las entradas de texto. Según el tipo de entrada, se visualizará un teclado u otro, y se permitirán por tanto unos caracteres u otros. En este caso, solo se permiten dígitos. A continuación se enumeran las más importantes, aunque existen otras muchas.

Nombre en el diseñador	Constante	Explicación
datetime	**TYPE_CLASS_DATETIME**	Fechas y horas.
date	**TYPE_DATETIME_VARIATION_DATE**	Fechas.
time	**TYPE_DATETIME_VARIATION_TIME**	Horas.
phone	**TYPE_CLASS_PHONE**	Teléfonos.
text	**TYPE_CLASS_TEXT**	Texto.
textUri	**TYPE_TEXT_VARIATION_URI**	Direcciones Web.
textCapWords	**TYPE_TEXT_FLAG_CAP_WORDS**	Texto con las iniciales de las palabras en mayúsculas.
textPassword	**TYPE_TEXT_VARIATION_PASSWORD**	Texto que es una contraseña.

number	**TYPE_CLASS_NUMBER**	Número.
numberSigned	**TYPE_NUMBER_FLAG_SIGNED**	Número con signo.
numberDecimal	**TYPE_NUMBER_FLAG_DECIMAL**	Número con parte decimal.

Finalmente, es necesario realizar algunas modificaciones para obtener un diseño más atractivo. Los dos **LinearLayout** horizontales podrían ocupar más espacio, de manera que los componentes no se apelotonaran en la parte de arriba de la pantalla. Sería posible especificar un tamaño en puntos independientes (las unidades con sufijo *dp*), pero esto lo haría poco portable a otros dispositivos. Es posible especificar, sin embargo, el porcentaje de pantalla a ocupar.

```
<LinearLayout
            android:layout_height="wrap_content" android:layout_gravity=”0.33”
            android:orientation="horizontal"
            android:layout_width="fill_parent">
```

De esta forma se indica que de alto el contenedor ocupa una tercera parte de la pantalla, lo cual implica especificar *layout_height* como *wrap_content* (ajustarse al contenido). Nótese que *layout_gravity* puede especificarse para cualquier componente, no solo para **LinearLayout**'s.

En cualquier caso, siempre se puede indicar en *layout_height* y *layout_width* o bien *wrap_content*, lo cual limitirá el componente al tamaño de lo que alberga, o bien *fill_parent*, que rellenará el resto del tamaño de la pantalla sin usar.

En cuanto al contenido de cualquier component (por ejemplo, una etiqueta como **TextView**), se puede indicar su posición relativa con *android:gravity*. Por ejemplo, se puede indicar *right* en *android:gravity* para **EditText**, de manera que lo que se introduzca esté alineado a la derecha.

El archivo final *activity_main.xml* resultante puede verse a continuación.

```
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:weightSum="1">
    <LinearLayout
            android:orientation="horizontal"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:layout_weight="0.30">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/label_enter_dni"
            android:layout_gravity="center_vertical" android:layout_weight="0.33"/>
        <EditText
            android:id="@+id/edDni"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:inputType="number" android:layout_gravity="center_vertical" android:gravity="right"
            android:hint="@string/dni_placeholder" android:layout_weight="0.67"/>
    </LinearLayout>
    <LinearLayout
            android:orientation="horizontal"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:layout_weight="0.30">
        <TextView
                android:id="@+id/lblResultado"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/label_result"
                android:textAppearance="?android:attr/textAppearanceLarge"
                android:layout_gravity="center_vertical"/>
        <TextView
            android:id="@+id/lblLetra"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/label_default_result"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:layout_gravity="center_vertical"
            android:gravity="right"/>
    </LinearLayout>
</LinearLayout>
```

7 Accediendo a los componentes desde el código

Una vez creada la interfaz, es necesario acceder a ciertos componentes: *edDni*, el **EditText** en donde el usuario introducirá su número de DNI, y *lblLetra*, el **TextView** donde aparecerá la letra correspondiente como resultado del cálculo. El cálculo se realizará siempre que se pulse cualquier tecla en el **EditText**, por lo que es necesario crear un evento que reaccione en este caso.

El método de la actividad empleado para localizar un determinado componente gráfico (vista, según la terminología de programación en Android) es *findViewById(id)*.

```
View view = findViewById( R.id.lblLetra );
```

En el ejemplo anterior, se localiza el **TextView** relativo a la letra del DNI. La clase **View** es la superclase que representa a todos los componentes gráficos. Es por tanto más útil hacer un cast a **TextView**.

```
TextView textView = (TextView) findViewById( R.id.lblLetra );
```

El identificador que permite recuperar el componente gráfico requerido es generado por el propio SDK de Android. En la carpeta *gen/* se generan una serie de constantes que son accesibles desde **R**. El identificador indicado para el campo *id* en el **TextView** creado para la interfaz será el que esté accesible desde R.id (en algunos casos puede ser necesario recompilar).

8 Programando eventos

Las vistas (*view*) o componentes gráficos lanzan eventos a los que es necesario dar respuesta. Por ejemplo, deseamos que al pulsar una tecla en el editor de texto, se calcule la letra del NIF, puesto que es probable que el número de DNI introducido haya cambiado. El momento ideal para enlazar los componentes con sus eventos es el método *onCreate()* de la actividad.

Siguiendo las guías de AWT de Java, los eventos se manejan creando *listeners* (literalmente, *escuchadores*), de distintos tipos. Así, se crea un objeto de una clase con un comportamiento determinado, como se puede ver a continuación.

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

    EditText edDni = (EditText) this.findViewById( R.id.edDni );
    edDni.setOnKeyListener( new OnKeyListener() {
                public boolean onKey(View arg0, int arg1, KeyEvent arg2) {
                       Main.this.calcula();
                       return false;
                }
    });
}
```

En este caso, es necesario crear un objeto de la clase **OnKeyListener**, especificando *onKey()* con el comportamiento deseado. En este caso, cuando se pulse una tecla, se va a ejecutar *calcula()*, que realiza el trabajo de localizar las vistas y llamar a la clase **CalculoLetraNif** para obtener la letra correspondiente. El método se lista a continuación.

```
protected void calcula()
{
      // Connect components
      EditText edDni = (EditText) this.findViewById( R.id.edDni );
      TextView lblLetra = (TextView) this.findViewById( R.id.lblLetra );

      // Reckon the letter and display it
      try {
             int dni = Integer.parseInt( edDni.getText().toString() );
             char letra = CalculoLetraNif.calculaLetraNif( dni );

             lblLetra.setText( Character.toString( letra ) );
      } catch(NumberFormatException fmt)
      {
             lblLetra.setText( R.string.label_default_result );
      }

      return;
}
```

Es interesante tener en cuenta que el método **EditText**.*getText()* no devuelve un objeto de la clase **String**, sino de la clase **Editable**. Es necesario llamar a *toString()* para obtener el contenido.

9 Referencias

- Documentación y recursos de Android para desarrolladores (accedido en sept. 2015) http://developer.android.com/
- Tutoriales oficiales de Android https://developer.android.com/training/
- Eventos https://developer.android.com/guide/topics/ui/ui-events.html
- Interfaz de usuario https://developer.android.com/guide/topics/ui/overview.html