Desarrollo para dispositivos móviles con Android: vista de lista.

Contenido

1 Introducción

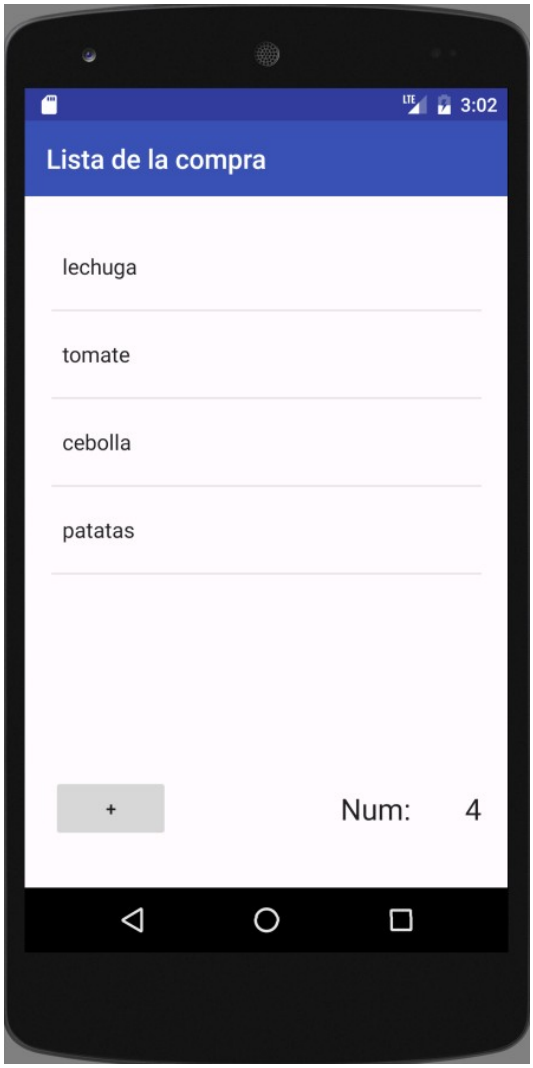
Es muy común mantener una serie de información en formato lista para que el usuario añada o elimine elementos. Ejemplos comunes son un *to-do* o una lista de la compra. A continuación se explica el uso de contenedores de lista a través de un ejemplo de lista de la compra.

Es importante distinguir entre el contenedor **ListView** y la actividad predefinida **ListActivity**. El primero es un contenedor que puede incluirse en cualquier *layout*, mientras la segunda es una actividad que únicamente contiene un widget **ListView**.

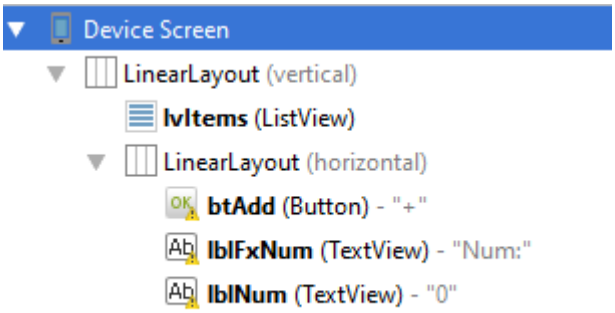
2 La App Lista de la compra

Sobre estas líneas puede verse la interfaz de la pequeña *app* dedicada a manejar la lista de la compra. Se trata de una interfaz muy sencilla, basada en una lista y botones.

2.1 ArrayAdapter



Antes de poder trabajar con el control **ListView**, es necesario comprender cómo funciona arquitecturalmente. El control tiene que reflejar el contenido de una colección de datos (típicamente un **ArrayList<>**, si bien puede ser también un array primitivo).



De lo que se trata es, básicamente, de separar modelo, vista y controlador. El modelo lo proporciona la colección de datos elegida por nosotros, mientras que *Adaptador* es el controlador. La vista, sin duda, es el componente **ListView**, que mostrará los datos por pantalla.

El adaptador más sencillo de todos los que se pueden utilizar es un **ArrayAdapter**. Se trata de un adaptador genérico que sirve, en el caso de las cadenas, tanto para un vector (**String[]**) como para una lista de cadenas (**ArrayList<String>**). En el siguiente ejemplo, código a colocar en *onCreate()*, se asume que existe un atributo privado ***ArrayList<String>*** *items*, y así mismo otro ***ArrayAdapter<String>*** *itemsAdapter*.

```
class MainActivity {
        private ArrayList<String> items;
        private ArrayAdapter<String> itemsAdapter;

        protected void onCreate(//...
        {
                //…
                // Link list view
                ListView lvItems = (ListView) this.findViewById( R.id.lvItems );

                // Pepare list and list adapter
                this.items = new ArrayList<String>();
                this.itemsAdapter = new ArrayAdapter<String>(
                        this,
                        android.R.layout.simple_selectable_list_item,
                        this.items );

                lvItems.setAdapter( this.itemsAdapter );

                //...
        }
}
```

Cuando se introduzca un elemento en el adaptador, mediante su método *add()*, el mismo hará dos cosas: llamar al mismo método *add()* en la colección, a la vez de avisar a **ListView** de que debe cambiar la apariencia. No siempre todos los métodos deseados están disponibles en el adaptador: así, por ejemplo, para eliminar un elemento dada su posición, el método *remove(i)* no está disponible en el adaptador, pero sí en la lista. En esos casos, se puede hacer la tarea de borrado en la colección misma, y después avisar del cambio con el método del adaptador *notifyDataSetChanged()*.

Por otra parte, cuando se crea un adaptador, se está poniendo en contacto, al fin y al cabo, la estructura de datos con la vista. Así, es necesario indicar el *layout* que se va a utilizar para cada elemento. Afortunadamente, existen unos cuantos *layouts* predeterminados que se pueden utilizar directamente, como *android.R.layout_simple_selectable_list_item*.

2.2 Manejando la lista

Una vez enlazada la estructura de datos con la vista, solamente queda manejarla. Si creamos un botón *btAdd* que permite añadir elementos a la lista, es posible crear nuevas posiciones fácilmente.

```
private void onAdd()
{
        final EditText edText = new EditText( this );
        final AlertDialog.Builder builder = new AlertDialog.Builder( this );

        builder.setTitle("A comprar...");
        builder.setMessage( "Nombre" );
        builder.setView( edText );
        builder.setPositiveButton( "+", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                final String text = edText.getText().toString();
                Main.this.itemsAdapter.add( text );
                Main.this.updateStatus();
            }
        });
```

```
        builder.setNegativeButton( "Cancel", null );
        builder.create().show();
}
```

Solo es necesario manejar el adaptador para añadir, eliminar u obtener elementos de la lista. Este adaptador se encarga de forma transparente de actualizar la lista con los cambios.

En cuanto a eliminar elementos, se empleará la capacidad *long click* (pulsar durante largo tiempo) para borrar un elemento. A continuación, se muestra código para el evento *onCreate()*, que habrá que añadir al ya existente.

```
class MainActivity {
        protected void onCreate(//…
        {
             // ...
             // Preparar el pulsar largo para borrar un elemento
             lvItems.setLongClickable( true );
             lvItems.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
               @Override
               public boolean onItemLongClick(AdapterView<?> adapterView, View view, int i, long l)
              {
                boolean toret = false;

                if ( i >= 0 ) {
                    Main.this.items.remove( i );
                    Main.this.itemsAdapter.notifyDataSetChanged();
                    Main.this.updateStatus();
                    toret = true;
                }

                return toret;
              }
             });

             // Preparar el botón para insertar en la lista
             btAdd.setOnClickListener(new View.OnClickListener() {
              @Override
              public void onClick(View view) {
                 MainActivity.this.onAdd();
              }
             });
}
```

3 La App *Burguer Builder*

En las siguientes secciones se va a utilizar una aplicación como ejemplo para desarrollar vistas **ListView** más complejas. Se trata de una app que permite elegir los ingredientes para una hamburguesa, a saber: lechuga, tomate, queso, york y cebolla. Cada uno de estos ingredientes tiene un coste asociado. También hay un grupo de ingredientes fijo, presentes en cualquier hamburguesa, que son el pan y la carne, con un coste de 1 y 2€ respectivamente.

A continuación, se incluye la lógica de negocio: una clase que calcula el coste de la hamburguesa según los ingredientes seleccionados. En cualquier versión de esta aplicación, residirá en el subpaquete (*subpackage*) *Core*.

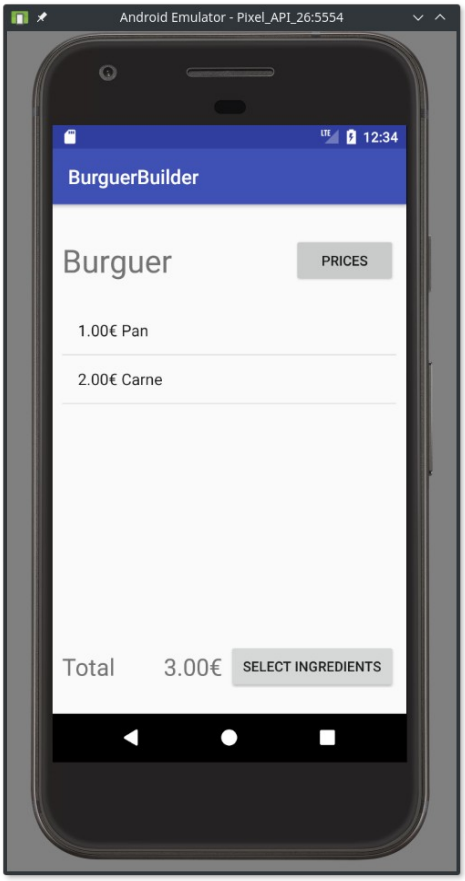
```
/** Calculates costs related to a burguer. */
public class BurguerConfigurator {
    public static final String DELIMITER = ", ";
    public static final double BASE_COST = 3.0;

    public BurguerConfigurator()
    {
        this.selected = new boolean[ INGREDIENTS.length ];

        assert INGREDIENTS.length != COSTS.length:
                "all arrays should have the same length";
    }

    /** Calculates the cost for the burguer
     * @return A real with the total cost.
     */
    public double calculateCost()
    {
        double toret = BASE_COST;

        for(int i = 0; i < this.selected.length; ++i) {
            if ( this.selected[ i ] ) {
                toret += COSTS[ i ];
            }
        }

        return toret;
    }

    /** Returns the nth global element given by selectedPos in the vector of vector of selections */
    public int getGlobalPosOfSelected(int selectedPos)
    {
        int toret = 0;

        for(int i = 0; i < this.selected.length; ++i) {
            if ( this.selected[ i ] ) {
                --selectedPos;
                if ( selectedPos < 0 ) {
                    toret = i;
                    break;
                }
            }
        }

        return toret;
    }

    /** Changes the ingredient set at position i with value. */
    public void setSelected(int i, boolean value)
    {
        this.selected[ i ] = value;
    }

    /** Returns the live boolean vector for ingredient selection */
    public boolean[] getSelected()
    {
        return Arrays.copyOf( this.selected, this.selected.length );
    }
```

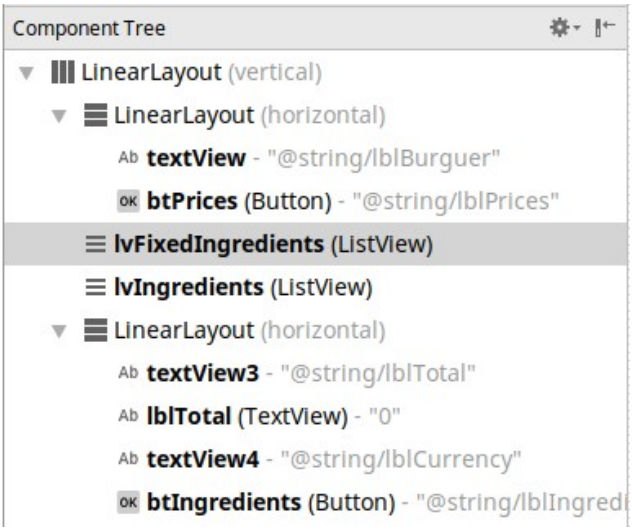
```
    @Override
    public String toString()
    {
        String delimiter = "";
        StringBuilder toret = new StringBuilder();

        for(int i = 0; i < this.selected.length; ++i) {
            if ( this.selected[ i ] ) {
                toret.append( delimiter );
                toret.append( INGREDIENTS[ i ] );
                delimiter = DELIMITER;
            }
        }

        return toret.toString();
    }

    /** Returns selected ingredients as a string.
     *  @param newDelimiter The delimiter to use to end each line.
     *  @returns A single string with the whole contents.*/
    public String toListWith(String newDelimiter)
    {
        return this.toString().replace( DELIMITER, newDelimiter );
    }

    /** Represents all available (selectable) ingredients */
    public static String[] INGREDIENTS = new String[] {
            "Lechuga",
            "Tomate",
            "Queso",
            "York",
            "Cebolla"
    };

    /** Parallel array to ingredients, representing their costs */
    public static double[] COSTS = new double[] {
            0.1,
            0.5,
            1,
            0.75,
            0.1
    };

    /** Represents all fixed, mandatory ingredients */
    public static String[] FIXED_INGREDIENTS = new String[] {
            "Pan",
            "Carne"
    };

    /** Parallel array to fixed ingredients, representing their costs */
    public static double[] FIXED_COSTS = new double[] {
            1,
            2
    };

    private boolean[] selected;
}
```

En cualquier versión de esta aplicación deberá mostrarse el coste fijo de la hamburguesa, así como el coste de cada uno de los ingredientes seleccionadas, y finalmente el coste total en la parte inferior. También debe ser posible conocer los costes de los ingredientes no seleccionados.

3.1 Creando un *ListView* simple

La primera aproximación a esta aplicación (*app* **BurguerBuilderListView**) es el uso de una **ListView** por defecto, en el que cada entrada de la lista es un **TextView** (es decir, texto simple). Así, la interfaz se divide, de arriba abajo, en un botón para ver todos los costes de todos los ingredientes, un **ListView** con los costes fijos que nunca cambia, un **ListView** con los ingredientes seleccionados, y una serie de etiquetas en el borde inferior que muestran el coste total, así como un botón para realizar la selección de ingredientes real.





A continuación, se incluye el listado del XML que representa la vista.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp"
    tools:context="com.devbaltasarq.burguerbuilderlistview.view.MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0.15"
        android:gravity="center_vertical">
```

```
        <TextView
            android:id="@+id/textView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical|center_horizontal"
            android:layout_weight="0.75"
            android:gravity="center_vertical"
            android:text="@string/lblBurguer"
            android:textSize="32sp" />
        <Button
            android:id="@+id/btPrices"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.25"
            android:text="@string/lblPrices" />
    </LinearLayout>
    <ListView
        android:id="@+id/lvFixedIngredients"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0.20"/>

    <ListView
        android:id="@+id/lvIngredients"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0.5"
        android:textColor="@android:color/black"
        android:textSize="22sp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_weight="0.15"
        android:gravity="center_vertical"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textView3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.2"
            android:text="@string/lblTotal"
            android:textSize="24sp" />

        <TextView
            android:id="@+id/lblTotal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.6"
            android:gravity="end"
            android:text="0"
            android:textAlignment="textEnd"
            android:textSize="24sp" />

        <TextView
            android:id="@+id/textView4"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.1"
            android:text="@string/lblCurrency"
            android:textSize="24sp" />

        <Button
            android:id="@+id/btIngredients"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.1"
            android:text="@string/lblIngredientSelection" />
    </LinearLayout>
</LinearLayout>
```

En el caso de esta aplicación, tan solo es necesaria la clase **MainActivity**, al emplear las posibilidades por defecto de **ListView**.

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

        // Create the burguer configuration
        this.cfgBurguer = new BurguerConfigurator();

        // Create callback for the button allowing to check prices
        Button btPrices = (Button) this.findViewById( R.id.btPrices );
        btPrices.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                MainActivity.this.showPricesDialog();
            }
        });

        // Create callback for the button allowing to select ingredients
        Button btIngredients = (Button) this.findViewById( R.id.btIngredients );
        btIngredients.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                MainActivity.this.showIngredientsDialog();
            }
        });

        this.showFixedIngredients();
        this.updateTotals();
    }

    //…
    private BurguerConfigurator cfgBurguer;
}
```

El método *onCreate()* establece la parte activa de la aplicación: dos botones, el que permitirá elegir los ingredientes, y el que mostrarás los precios de todos y cada uno de los ingredientes. El siguiente paso es ser capaz de mostrar la lista de ingredientes fijos, no seleccionables.

```
 public class MainActivity //...
    private void showFixedIngredients()
    {
        ListView lvFixedIngredients = (ListView) this.findViewById( R.id.lvFixedIngredients );

        String[] fixedIngredients = new String[] {
                String.format( "%4.2f€ %s", BurguerConfigurator.FIXED_COSTS[ 0 ],
                        BurguerConfigurator.FIXED_INGREDIENTS[ 0 ] ),
                String.format( "%4.2f€ %s", BurguerConfigurator.FIXED_COSTS[ 1 ],
                        BurguerConfigurator.FIXED_INGREDIENTS[ 1 ] )
        };

        lvFixedIngredients.setAdapter(
                new ArrayAdapter<String>(
                        this,
                        android.R.layout.simple_list_item_1,
                        fixedIngredients ) );

    }
    //...
}
```

Mostrar los ingredientes fijos es muy simple, pues se trata al fin y al cabo de un **ListView** que no aceptará ningún tipo de interacción.

```
public class MainActivity //...
    private void showPricesDialog()
    {
        final int NUM_INGREDIENTS = BurguerConfigurator.INGREDIENTS.length;
        final String[] ingredientsWithPrices = new String[ NUM_INGREDIENTS ];
        final TextView lblData = new TextView( this );
        AlertDialog.Builder dlg = new AlertDialog.Builder( this );
        dlg.setTitle( this.getResources().getString( R.string.lblPrices) );

        // Build list with prices
        for(int i = 0; i < ingredientsWithPrices.length; ++i) {
            ingredientsWithPrices[ i ] = String.format( "%4.2f€ %s",
                    BurguerConfigurator.COSTS[ i ],
                    BurguerConfigurator.INGREDIENTS[ i ] );
        }

        lblData.setText( String.join( "\n", ingredientsWithPrices ) );
        lblData.setPadding( 10, 10, 10, 10 );
        dlg.setView( lblData );
        dlg.setPositiveButton( "Ok", null );
        dlg.create().show();
    }
}
```

De nuevo no existe ninguna posibilidad de interacción con este diálogo, pues solo muestra los precios a título meramente informativo. De hecho, la parte interactiva es la que se muestra a continuación. Se trata de la actualización del **ListView** de ingredientes según las selecciones del usuario, y la actualización del **TextView** donde se muestra el total del coste.

```
public class MainActivity //…
    //...
    private void showIngredients()
    {
        final int NUM_ITEMS = this.cfgBurguer.getSelected().length;
        final boolean[] selections = this.cfgBurguer.getSelected();
        final ListView lvIngredients = (ListView) this.findViewById( R.id.lvIngredients );

        // Create list
        ArrayList<String> ingredients = new ArrayList<>();
        for(int i = 0; i < NUM_ITEMS; ++i) {
            if ( selections[ i ] ) {
                ingredients.add(
                        String.format( "%4.2f€ %s", BurguerConfigurator.COSTS[ i ],
                                        BurguerConfigurator.INGREDIENTS[ i ] ) );
            }
        }

        lvIngredients.setAdapter(
                new ArrayAdapter<String>(
                        this,
                        android.R.layout.simple_list_item_1,
                        ingredients ) );

        lvIngredients.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
            @Override
            public boolean onItemLongClick(AdapterView<?> adapterView, View view, int i, long l) {
                final int listPos = MainActivity.this.cfgBurguer.getGlobalPosOfSelected( i );

                MainActivity.this.cfgBurguer.setSelected( listPos, false );
                MainActivity.this.showIngredients();
                MainActivity.this.updateTotals();
                return true;
            }
        });
    }

    private void updateTotals()
    {
        final TextView lblTotal = (TextView) this.findViewById( R.id.lblTotal );

        lblTotal.setText(
                String.format( "%4.2f", MainActivity.this.cfgBurguer.calculateCost() ) );
    }

    //...
}
```

La única dificultad es que, al no mostrarse la lista de ingredientes completa, sino tan solo aquellos ya seleccionados, al eliminar (pulsación larga) un elemento, es necesario convertir la posición en la lista de elementos seleccionados en la posición de la lista global de ingredientes. Por ejemplo, si están seleccionados el tomate, la lechuga y la cebolla (el vector selected es *true*, *true*, *false*, *false*, *true*), un intento de eliminar la cebolla nos dará un resultado de 2, mientras que en realidad es la posición número 4 en el vector global de ingredientes. La responsabilidad de esta conversión reside en el método **BurguerConfigurator**.*getGlobalPosOfSelected(i)*, de manera que en el ejemplo anterior se le pasaría 2 (*i* sería 2), y devolvería 5, que es el índice del tercer ingrediente seleccionado en el **ListView**. La forma de hacerlo es simplemente contar el número de elementos seleccionados a medida que se recorre el vector de ingredientes, hasta que el conteo llega al índice seleccionado.

3.2 Creando un *ListView* con un *layout* propio para cada entrada (I)

El caso más típico es desear tener un una vista de lista con un *layout* específico para sus entradas (app **ListaCompraLayoutPropio**). Para comprender mejor el concepto, revisitaremos la aplicación de la lista de la compra. En esta ocasión, utilizaremos una clase **Item** que representará los productos a comprar, junto con el supermercado en el que los compraremos.

```
public class Item {
    public Item(String n, String s)
    {
        this.nombre = n;
        this.supermercado = s;
    }

    public String getNombre()
    {
        return this.nombre;
    }

    public String getSupermercado()
    {
        return this.supermercado;
    }

    public String toString()
    {
        final String SUPERMERCADO = this.getSupermercado();
        String toret;

        if ( SUPERMERCADO != null
          && !SUPERMERCADO.isEmpty() )
        {
            toret = String.format( "%s (%s)", this.getNombre(), this.getSupermercado() );
        } else {
            toret = this.getNombre();
        }

        return toret;
    }

    private String nombre;
    private String supermercado;
}
```

Sería posible utilizar un **ArrayList<Item>**, como ya hemos visto un poco más arriba con la aplicación del configurador de hamburguesas. Supongamos que deseamos conseguir un leve efecto: que el texto correspondiente al supermercado de cada ítem aparezca en negrita.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_vertical">

    <TextView   android:id="@+id/lblSupermercado"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Large"
        android:layout_weight=".4"
        android:textStyle="bold"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView   android:id="@+id/lblNombre"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
        android:gravity="center_horizontal"
        android:layout_marginStart="10dp"
        android:layout_marginLeft="10dp"
        android:layout_weight=".6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```

Se trata simplemente de un *layout* horizontal, que contiene dos **TextView**: uno lo utilizaremos para el supermercado, y el otro para el ítem en sí. Para crear una nueva *layout* desplegamos *res* en la vista del proyecto, y a continuación pulsamos el botón derecho en *layout*, seleccionando *new*, y a continuación *layout resource file*. Le llamaremos *listview_item* (escogemos un nombre muy detallado para el caso en el que empleemos diferentes *layouts* y de esta forma no dar opción a confusiones).

A continuación, debemos crear una clase que derivará de **ArrayAdapter**, y que indicará a Android cómo debe realizar la presentación de cada ítem, enlazando los datos con el *layout*.

La clave por tanto en la clase siguiente, ItemArrayAdapter, está en el método **ItemArrayAdapter**.*getView(p, v, vg)*, donde *p* es el número de fila, *v* la vista correspondiente a esa fila, y *vg* es el contenedor de todas las filas. Cuando creamos este adaptador, le pasamos el contexto y los ítems, de manera que los tendremos disponibles durante el método *getView()*, a través de los métodos **ArrayAdapter**.*getItem(i)* (que devuelve el elemento en la posición *i)*, y **ArrayAdapter**.*getContext()*, que devuelve el contexto que se le pasó al ser creado.

Así, los pasos a seguir en el método **ItemArrayAdapter**.*getView()* son los siguientes:

1. Determinar si es necesario crear la vista para la fila. La vista que nos pasa puede estar ya creada de antes, o puede ser *null*. En este último caso, es necesario crear la fila.
2. Crear la fila (si se ha determinado que es necesario), utilizando el inflador de *layouts* del contexto.
3. Asignar a las *views* dentro del *layout* los datos correspondientes a esa fila.

```
public class ItemArrayAdapter extends ArrayAdapter {
    public ItemArrayAdapter(Context context, ArrayList<Item> items)
    {
        super( context, 0, items );
    }

    @Override
    public View getView(int position, View view, ViewGroup parent)
    {
        final Context CONTEXT = this.getContext();
        final LayoutInflater INFLATER = LayoutInflater.from( CONTEXT );
        final Item ITEM = (Item) this.getItem( position );

        // Crear la vista si no existe
        if ( view == null ) {
            view = INFLATER.inflate( R.layout.listview_item, null );
        }

        // Rellenar los datos
        final TextView lblNombre = view.findViewById( R.id.lblNombre );
        final TextView lblSupermercado = view.findViewById( R.id.lblSupermercado );

        lblNombre.setText( ITEM.getNombre() );
        lblSupermercado.setText( ITEM.getSupermercado() );

        return view;
    }
}
```

Como se puede ver justo encima, se trata de utilizar el inflador de vistas (**LayoutInflater**), para crear un objeto **View** a partir de un layout determinado. Las siguientes veces que se llame a *getView()* ya se habrá creado esa fila, por lo que no será necesario volver a crearla. Finalmente, con **View**.*findViewById(s)*, se localizan las vistas correspondientes, y se les asigna valores.

Ya solo es necesario utilizar esta clase **ItemArrayAdapter** en lugar de **ArrayAdapter<Item>**, que es la que hemos utilizado hasta ahora. Para ello, empleamos el método **MainActivity**.*creaLista()*:

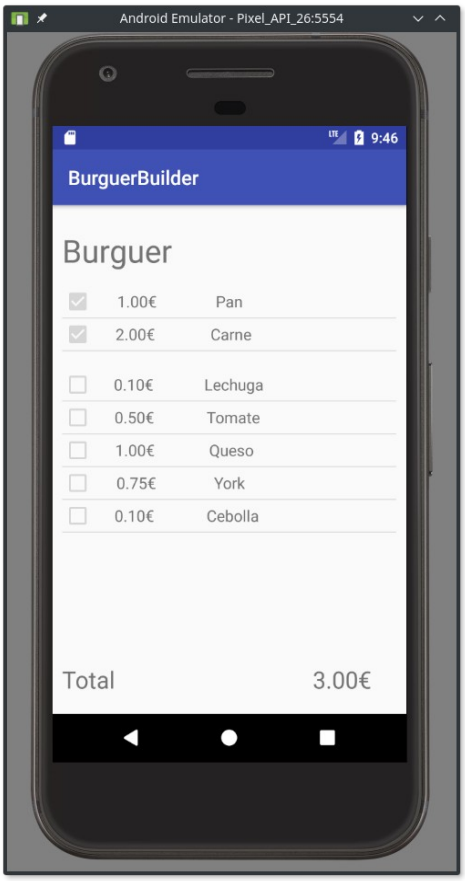
```
class MainActivity {

	// Más cosas…

	private void creaLista()
	{
		final ListView lvItems = this.findViewById( R.id.lvItems );

		this.adapterList = new ItemArrayAdapter( this, this.items );
		lvItems.setAdapter( this.adapterList );
	}
}
```

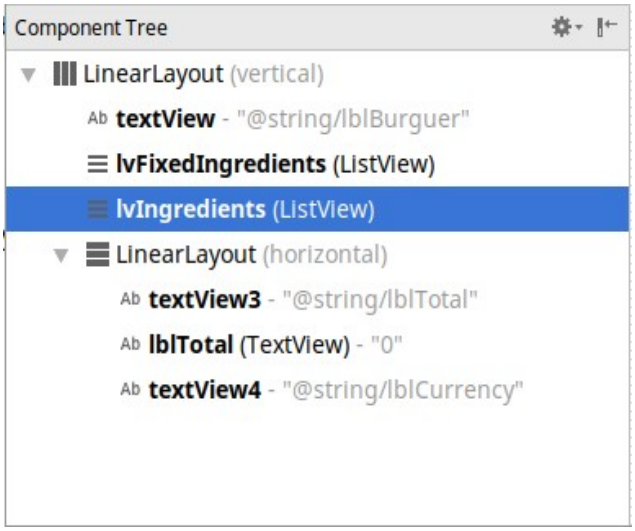
Este método es llamado desde **MainActivity**.*onCreate()*, de manera que la lista queda ya construida, y cualquier cambio a través del adaptador (nuestra clase **ItemArrayAdapter**, que hereda todos los métodos de **ArrayAdapter<Item>**), se reflejará inmediatamente en ella.

3.3 Creando un *ListView* con un *layout* propio para cada entrada (II)

Para esta aplicación (*app* **BurguerBuilderComplexListView**), se empleará un **ListView** personalizado. Aunque una lista que muestra texto es probablemente la más utilizada, existen otros casos en los que es conveniente mostrar, además de texto (o en su lugar), un icono, o quizás, como en el caso que nos ocupa, un cuadro de verificación (*check box*) al lado de un texto.

El ejercicio en el que se va a basar la explicación sobre cómo crear listas con un *layout* propio es *Burguer Builder*, una aplicación diseñada para calcular el coste de una hamburguesa según los ingredientes que elijamos.





Como se puede ver sobre estas líneas, el pan y la carne están elegidos por defecto, de manera que suponen el coste básico de la hamburguesa 3€. El resto son ingredientes que se pueden activar o desactivar.

El pan y la carne se indican mediante un **ListView** decorativo (*lvFixedIngredients*), de manera que solo muestra las dos líneas con esos ingredientes a título informativo. El **ListView** *lvIngredients* es el que se encarga de manejar los ingredientes que pueden ser activados y desactivados, y el que recibirá la atención en esta sección. En la parte inferior, tres *TextView* se encargan de mostrar el texto fijo “Total”, el precio calculado hasta el momento de la hamburguesa, y el texto fijo “€”, para el caso en el que en el futuro se quisieran soportar otras monedas. De todos ellos, el central, *lblTotal*, es el único que se manejará para mostrar el cálculo del coste de los ingredientes opcionales.

El código fuente del XML para al vista se muestra a continuación.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp"
    tools:context="com.devbaltasarq.burguerbuildercomplexlistview.view.MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical|center_horizontal"
        android:layout_weight="0.1"
        android:gravity="center_vertical"
        android:text="@string/lblBurguer"
        android:textSize="32sp" />

    <ListView
        android:id="@+id/lvFixedIngredients"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0.25" />

    <ListView
        android:id="@+id/lvIngredients"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0.5"
        android:textColor="@android:color/black"
        android:textSize="22sp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_weight="0.15"
        android:gravity="center_vertical"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textView3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.2"
            android:text="@string/lblTotal"
            android:textSize="24sp" />

        <TextView
            android:id="@+id/lblTotal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.7"
            android:gravity="end"
            android:text="0"
            android:textAlignment="textEnd"
            android:textSize="24sp" />

        <TextView
            android:id="@+id/textView4"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.1"
            android:text="@string/lblCurrency"
            android:textSize="24sp" />
    </LinearLayout>
</LinearLayout>
```

Los vectores de ingredientes INGREDIENTS, COST y selected son paralelos: la posición 0 de *INGREDIENTS* nos da el nombre del ingrediente (lechuga), la posición 0 del vector *COST* nos da su coste (0.1), y finalmente, la posición 0 de *selected* nos indica si se ha seleccionado o no. Mientras que los dos primeros son vectores de constantes estáticos (asociados a la clase), el último es un atributo (es decir, asociado a la instancia). En este ejemplo, se considera que los precios y los

nombres de los ingredientes no van a cambiar con frecuencia. El método *calculateCost()* es el que se encarga de recorrer el vector selected e ir sumando las posiciones correspondientes en el vector *COST*, obteniendo el coste total.

En cuanto a la clase **MainActivity**, que engloba el manejo de la vista y los controladores, se escribiría de esta forma:

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

        // Create the burguer configuration
        this.cfgBurguer = new BurguerConfigurator();

        // Show components and initial total
        this.showFixedIngredients();
        this.showIngredients();
        this.updateTotals();
    }

    //...
    private void updateTotals()
    {
        final TextView lblTotal = (TextView) this.findViewById( R.id.lblTotal );

        lblTotal.setText(
                String.format( "%4.2f", MainActivity.this.cfgBurguer.calculateCost() ) );
    }

    private BurguerConfigurator cfgBurguer;
    //...
}
```

De la clase del *Core* **BurguerConfigurator** se crea una instancia que será la que lleve el control de los ingredientes seleccionados o no. En cuanto al tercio superior de la actividad de la aplicación que estará ocupado por los ingredientes fijos, se encarga el método *showFixedIngredients()*, al que solo se llama una vez. En cuanto al *layout* para cada uno de los elementos de la lista, se emplea *android.R.layout.simple_list_item1*, como en el caso anterior. Nótese que este **ListView** solo se emplea para que el usuario conozca aquellos ingredientes que están ya incluidos: en realidad, no se interaccione desde este **ListView** con el usuario, es solo informativo. Así, el adaptador es un **ArrayAdapter<String>** como en el caso anterior, que en este caso actúa sobre un vector de cadenas fijo de dos elementos (pan y carne).

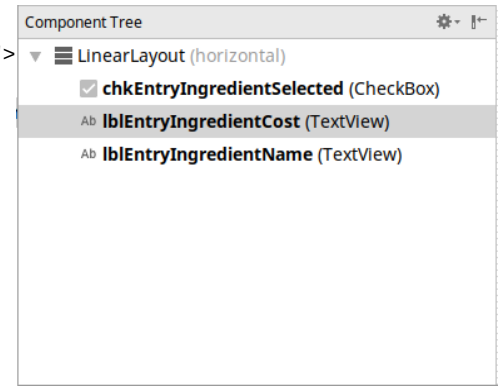
En el caso de *updateTotals()*, se trata del método que será llamado cada vez que la lista de ingredientes cambie (en el sentido de que un ingrediente sea acivado o desactivado, pues el número de elementos de la lista permanecerá fijo), calculando el coste del producto a travñes de **BurguerConfigurator**.*calculateCost()* y mostrando el resultado en *lblTotal*.

Lo más interesante de esta aplicación es el **ListView** *lvIngredients*, del que se tratará en el resto de la sección. Para empezar, es necesario indicar la forma en la que se mostrará la información en cada entrada del **ListView**, y para ello desde Android se proporciona la posibilidad de crear nuestro propio **Layout** (en lugar de utilizar uno predefinido como hicimos anteriormente). En este caso, deseamos tener en horizontal: un cuadro de verificación (*checkbox*, que indique si el ingrediente está activado o no), una etiqueta (para el precio del ingrediente), y finalmente otra etiqueta (para el nombre del ingrediente).

Para crear una nueva *layout* desplegamos *res* en la vista del proyecto, y a continuación

pulsamos el botón derecho en *layout*, seleccionando *new*, y a continuación *layout resource file*. Le llamaremos *listview_ingredient_entry* (escogemos un nombre muy detallado para el caso en el que empleemos diferentes *layouts* y de esta forma no dar opción a confusiones). El código en su interior aparece a continuación.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_vertical"
    android:descendantFocusability="blocksDescendants">

    <CheckBox
        android:id="@+id/chkEntryIngredientSelected"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="0.1"
        android:enabled="false"/>

    <TextView
        android:id="@+id/lblEntryIngredientPrice"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="0.25"
        android:gravity="center_vertical"
        android:textSize="16sp"/>

    <TextView
        android:id="@+id/lblEntryIngredientName"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="0.65"
        android:gravity="center_vertical"
        android:textSize="16sp"/>
</LinearLayout>
```



En este *layout* se crean tres componentes dentro de un **LinearLayout** horizontal: un **Checkbox**: *chkEntryIngredientSelected*, un **TextView**: *lblEntryIngredientCost*, y y finalmente otro **TextView**: *lblEntryIngredientName*.

Ahora es necesario crear una clase que represente a los objetos que se van a utilizar para cada una de las entradas de la lista: deben representar la información de cada ingrediente con la posibilidad de selección en el caso de los ingredientes seleccionables (los normales) y los fijos (el pan y la carne). En este caso, se crea una nueva clase que soporta la interfaz común para ambos: **ListViewIngredientEntry**.

```
/** An entry in a ListView for an ingredient */
public abstract class ListViewIngredientEntry {
    /** @return Whether this ingredient has been selected or not */
    public abstract boolean isSelected();

    /** Changes the selection of the ingredient to the given value */
    public abstract void setSelected(boolean selected);

    /** Inverts the current selection state */
    public void invertSelection()
    {
        this.setSelected( !this.isSelected() );
    }

    /** @returns the individual cost of this ingredient. */
    public abstract double getCost();


    /** @returns the name of this ingredient */
    public abstract String getIngredient();
}
```

Los ingredientes normales, es decir, que pueden ser seleccionados, deben ser conectados con

una posición concreta dentro del vector de ingredientes seleccionados, de lo cual se encarga **ListViewRegularIngredientEntry**, una especialización de la anterior, **ListViewIngredientEntry**.

```
/** Adapts each entry in the ListView to the burguer configurator object: selected, name and cost */
public class ListViewRegularIngredientEntry extends ListViewIngredientEntry {
    /** Connects this entry with an ingredient position in the burguer configurator list. */
    public ListViewEntry(BurguerConfigurator bc, int pos)
    {
        this.bc = bc;
        this.pos = pos;
    }

    /** @return The position of this ingredient in the burguer configurator list */
    public int getPos()
    {
        return this.pos;
    }

    /** @return Whether this ingredient has been selected or not */
    public boolean isSelected() {
        return this.bc.getSelected()[ this.pos ];
    }

    /** Changes the selection of the ingredient to the given value */
    public void setSelected(boolean selected) {
        this.bc.setSelected( this.pos, selected );
    }

    /** Inverts the current selection state */
    public void invertSelection()
    {
        this.setSelected( !this.isSelected() );
    }

    /** @returns the individual cost of this ingredient. */
    public double getCost() {
        return BurguerConfigurator.COSTS[ this.pos ];
    }


    /** @returns the name of this ingredient */
    public String getIngredient() {
        return BurguerConfigurator.INGREDIENTS[ this.pos ];
    }

    private BurguerConfigurator bc;
    private int pos;
}
```

Aquellos ingredientes que son fijos son más sencillos: no se pueden desmarcar (en realidad, con no definir un *listener* para el click del ratón sería suficiente), y devuelve un ingrediente y coste fijos. Se trata de la clase **ListViewFixedIngredientEntry**, también una especialización de **ListViewIngredientEntry**.

```
/** Represents a fixed ingredient (not selectable) entry in a list view. */
public class ListViewFixedIngredientEntry extends ListViewIngredientEntry{
    public ListViewFixedIngredientEntry(String ingredient, double cost)
    {
        this.ingredient = ingredient;
        this.cost = cost;
    }

    /** @return Whether this ingredient has been selected or not */
    public boolean isSelected() {
        return true;
    }

    /** Changes the selection of the ingredient to the given value */
    public void setSelected(boolean selected) {}

    /** @returns the individual cost of this ingredient. */
    public double getCost() {
        return this.cost;
    }
```

```
    /** @returns the name of this ingredient */
    public String getIngredient() {
        return this.ingredient;
    }

    private String ingredient;
    private double cost;
}
```

Es necesario ahora tener un adaptador que haga que el **ListView** represente para cada entrada el objeto **ListViewIngredientEntry** en el **Layout** *listview_ingredient_entry*, correspondiente a cada ingrediente. Este es el papel que normalmente hace **ArrayAdapter<>**, pero que tenemos que detallar un tanto, en conreto en cuanto al método *getView(position, converView, parent)*, que es el que se encarga de hacer esto.

```
/** Explains to the ListView how to show each entry. */
public class ListViewIngredientEntryArrayAdapter extends ArrayAdapter<ListViewIngredientEntry> {
    public ListViewIngredientEntryArrayAdapter(Context context, ListViewIngredientEntry[] entries)
    {
        super( context, 0, entries );

    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent)
    {
        final LayoutInflater layoutInflater = LayoutInflater.from( this.getContext() );
        final ListViewIngredientEntry entry = this.getItem( position );

        if ( convertView == null ) {
            convertView = layoutInflater.inflate( R.layout.listview_ingredient_entry, null );
        }

        final CheckBox chkSelected = convertView.findViewById( R.id.chkEntryIngredientSelected );
        final TextView lblIngredient = convertView.findViewById( R.id.lblEntryIngredientName );
        final TextView lblCost = convertView.findViewById( R.id.lblEntryIngredientCost);

        chkSelected.setChecked( entry.isSelected() );
        lblIngredient.setText( entry.getIngredient() );
        lblCost.setText( String.format( "%4.2f€", entry.getCost() ) );

        return convertView;
    }
}
```

El método *getView(position, converView, parent)*, recibe el objeto **View** que representa el dibujado de la entrada númer *position*. Si este objeto es *null*, significa que todavía no se ha creado, de manera que es necesario crear este objeto a partir del **Layout** *listview_ingredient_entry*, accesible a través de *R.layout.listview_ingredient_entry*, utilizando un objeto **LayoutInflater**. Todas las actividades (**Activity**) cuentan con un *inflador*, accesible a través del método estático **LayoutInflater**.*from(context)*. Una vez obtenido el *inflador*, se llama al método *inflate(l)* con el layout deseado *l* (*R.layout.listview_ingredient_entry* en nuestro caso), de manera que ya sea posible acceder a los **TextView**, y **CheckBox** que contiene (a través del método, ya conocido *findViewById(id)*). Para la posición, obtenemos el objeto **ListViewEntry** correspondiente empleando el método *getItem(i)*. Se rellena **View** convertView con los datos necesarios, y se devuelve, con lo que el **ListView** solo tiene que visualizarlo para representar la entrada número *position*.

Solo falta ahora crear el **ListView** de la forma adecuada en **MainActivity**. Se trata, en concreto, del método *showIngredientList()*, llamado una sola vez, desde **MainActivity**.*onCreate()*.

```
public class MainActivity //… {
    //…
    private void showIngredients()
    {
        final int NUM_INGREDIENTS = this.cfgBurguer.getSelected().length;
        final ListView lvIngredients = (ListView) this.findViewById( R.id.lvIngredients );

        // Create list
        ListViewIngredientEntry[] ingredientEntryAdapterList =
                                    new ListViewRegularIngredientEntry[ NUM_INGREDIENTS ];

        for(int i = 0; i < NUM_INGREDIENTS; ++i) {
            ingredientEntryAdapterList[ i ] =
                                    new ListViewRegularIngredientEntry( this.cfgBurguer, i );
        }

        this.ingredientAdapterList = new ListViewIngredientEntryArrayAdapter(
                this,
                ingredientEntryAdapterList
        );

        lvIngredients.setAdapter( this.ingredientAdapterList );

        lvIngredients.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
                MainActivity.this.ingredientAdapterList.getItem( i ).invertSelection();
                MainActivity.this.ingredientAdapterList.notifyDataSetChanged();
                MainActivity.this.updateTotals();
            }
        });
    }

    private BurguerConfigurator cfgBurguer;
    private ListViewIngredientEntryArrayAdapter ingredientAdapterList;
}
```

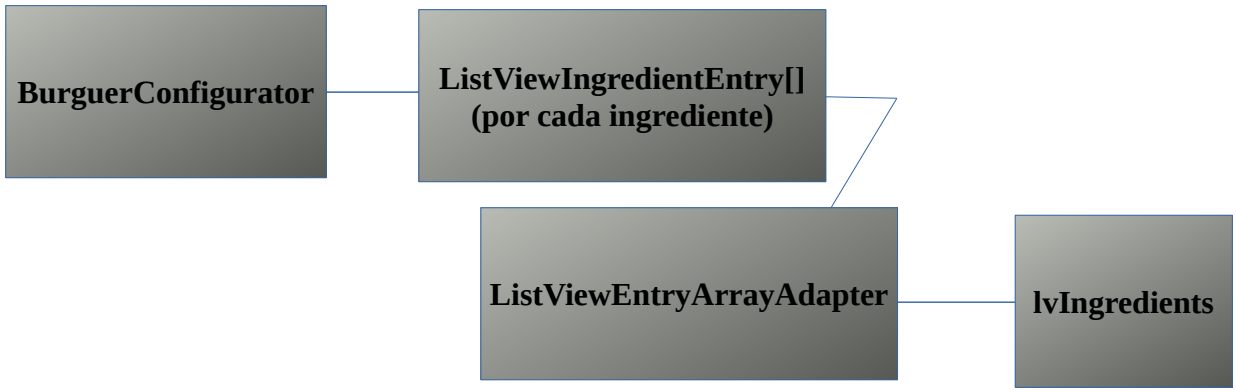
De forma similar, es necesario crear el método *showFixedIngredientList()*, llamado también una sola vez, desde **MainActivity**.*onCreate()*. Al no necesitar un *listener*, es mucho más simple que el anterior.

```
public class MainActivity //…
    private void showFixedIngredients()
    {
        ListView lvFixedIngredients = (ListView) this.findViewById( R.id.lvFixedIngredients );

        ListViewIngredientEntry[] fixedIngredientsList = new ListViewFixedIngredientEntry[] {
                new ListViewFixedIngredientEntry(
                BurguerConfigurator.FIXED_INGREDIENTS[ 0 ], BurguerConfigurator.FIXED_COSTS[ 0 ] ),
                new ListViewFixedIngredientEntry(
                BurguerConfigurator.FIXED_INGREDIENTS[ 1 ], BurguerConfigurator.FIXED_COSTS[ 1 ] )
        };

        // Create list
        ListViewIngredientEntryArrayAdapter fixedIngredientEntryAdapterList =
                new ListViewIngredientEntryArrayAdapter( this, fixedIngredientsList );

        lvFixedIngredients.setAdapter( fixedIngredientEntryAdapterList );
    }
```

```
    private void showFixedIngredients()
    {
        ListView lvFixedIngredients = (ListView) this.findViewById( R.id.lvFixedIngredients );

        ListViewIngredientEntry[] fixedIngredientsList = new ListViewFixedIngredientEntry[ 2 ];
        fixedIngredientsList[ 0 ] = new ListViewFixedIngredientEntry( "Pan", 1.0 );
        fixedIngredientsList[ 1 ] = new ListViewFixedIngredientEntry( "Carne", 2.0 );

        // Create list
        ListViewIngredientEntryArrayAdapter fixedIngredientEntryAdapterList =
                new ListViewIngredientEntryArrayAdapter( this, fixedIngredientsList );


        lvFixedIngredients.setAdapter( fixedIngredientEntryAdapterList );
    }
    //…
}
```

Es necesario llamar explícitamente a **ArrayAdapter<>**.*notifyDataSetChanged()*, ya que no estamos eliminando o añadiendo ningún elemento, por lo que el adaptador no se da por enterado de que algo haya cambiado.

De esta manera, el propio **ListView** es el que se encarga de gestionar toda la funcionalidad de la parte visual de la aplicación. En el diagrama adjunto se muestra la secuencia de adaptadores necesaria para poder hacer que **ListView** visualice los datos contenidos en **BurguerConfigurator**.



4 Referencias

- Documentación y recursos de Android para desarrolladores (accedido en sept. 2015) http://developer.android.com/
- Android: list view http://developer.android.com/reference/android/widget/ListView.html
- Tutoriales: list view activity http://developer.android.com/guide/topics/ui/layout/listview.html
- Ejemplo de la lista de la compra: http://github.com/Baltasarq/ListaCompra
- Ejemplo de la lista de la compra con un ListView complejo: https://github.com/Baltasarq/ListaCompraLayoutPropio/
- Ejemplo *BurguerBuilder* con un *ListView* simple: https://github.com/Baltasarq/BurguerBuilderListView
- Ejemplo *BurguerBuilder* con un *ListView* complejo: https://github.com/Baltasarq/BurguerBuilderComplexListView