# Menús, listas y barras de progreso

## Índice

En esta sesión seguimos hablando de interfaces gráficas en Android, presentando algunos elementos que requieren un estudio especial y un poco más profundo.

# 1. Barras de progreso

Una de las características esenciales que debe presentar cualquier aplicación móvil es la ausencia de latencia a la hora de responder al usuario. Debemos evitar que el tiempo de espera entre que el usuario interaccione con la aplicación y obtenga su respuesta sea demasiado alto. Una alta latencia podría incluso podrucir el cierre de la aplicación.

La forma más adecuada de evitar que se produzca latencia, sobre todo a la hora de realizar conexiones de red, es el uso de `AsyncTasks`. Este concepto será explorado en futuros módulos del curso. Otra posibilidad, que es la que vamos a explorar en esta sesión, es utilizar barras de progreso para informar al usuario del tiempo restante para completar una determinada operación.

En esta sección veremos dos formas posibles de incorporar un componente de este tipo a nuestra aplicación.

## 1.1. Barra de progreso circular

Utilizaremos esta barra de progreso cuando el tiempo que requerirá una tarea es indeterminado o desconocido a priori. La aplicación mostrará un elemento gráfico animado. Simplemente lo incorporamos al layout y lo mostramos cuando sea necesario. En el layout podríamos añadir un elemento `ProgressBar` de la siguiente forma:

```
<ProgressBar
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/progressbar"
/>
```

Una vez añadido al layout en la posición adecuada tan solo será cuestión de mostarlo u ocultarlo cuando sea necesario. Para ello usamos el método `setVisibility`, pasando como parámetro el valor `View.VISIBLE` o `View.INVISIBLE`.

Barra de progreso circular

## 1.2. Barra de progreso lineal

Al contrario que en el caso de la barra de progreso circular, la barra de progreso lineal será utilizada cuando sí conozcamos el tiempo o número de pasos que va a requerir la finalización de una determinada tarea. Como el objetivo va a ser que la barra de progreso

se vaya actualizando conforme se va avanzando en la tarea a completar, deberemos incorporar algún mecanismo para ir modificando el estado del componente gráfico. Esto se hará mediante hilos.

Podemos incorporar una barra de progreso lineal a un layout mediante un elemento de tipo `ProgressBar` a cuyo atributo `style` le hemos asignado el valor que se puede ver en el siguiente ejemplo:

```
<ProgressBar
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    style="?android:attr/progressBarStyleHorizontal"
    android:id="@+id/progreso"
    android:max="100"
/>
```

El atributo `android:max` establece, evidentemente, el máximo valor que puede tomar la barra de progreso. Ten en cuenta que no existe un atributo para establecer el valor mínimo. El valor inicial será por lo tanto siempre 0. Si queremos un rango de valores con otro valor inicial, deberemos sumar siempre ese valor inicial a cualquier valor leído desde el `ProgressBar`.

A continuación se muestra un ejemplo muy sencillo que ilustra el uso del `ProgressBar` lineal:

```
public class AndroidProgressBar extends Activity {

        ProgressBar progreso;
        int miProgreso = 0;

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

                progreso=(ProgressBar)findViewById(R.id.progreso);

                new Thread(myThread).start();
            }

        private Runnable myThread = new Runnable(){

                  public void run() {
                           while (miProgreso<100){
                            try{
myHandle.sendMessage(myHandle.obtainMessage());
                                     Thread.sleep(1000);
                            } catch(Throwable t){}
                           }
                }

                Handler myHandle = new Handler(){
                      @Override
                         public void handleMessage(Message msg) {
                          miProgreso++;
                          progreso.setProgress(miProgreso);
                         }
                 };
```

```
    };
}
```

Como se puede observar, la actualización de la barra de progreso se realiza mediante una instancia de `Runnable`. Esto crea un hilo aparte que se encarga de dicha actualización mientras se realizan otras tareas. En este caso la actividad no hace nada; simplemente hace una llamada a `Thread.sleep` para ir aumentando el valor de la barra de progreso de uno en uno cada segundo.

Barra de progreso lineal

## 1.3. SeekBar

Un `SeekBar` es un tipo de vista que permite seleccionar un valor entero como entrada para una determinada actividad. Se trata de una extensión del `ProgressBar` a la que se le ha añadido un control arrastable con el que el usuario puede interaccionar. El valor asociado al `SeekBar` dependerá de la posición en la que se encuentre dicho control.

Para incorporar un `SeekBar` a nuestro layout podemos añadir el siguiente código:

```
<SeekBar android:id="@+id/seek"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="2" />
```

donde `android:max` indica el valor máximo, como en el caso de un `ProgressBar`. También, como en ese caso, no es posible establecer un valor mínimo; éste será siempre 0. El atributo `android:progress` proviene de la clase `ProgressBar` y establece el incremento en el valor representado por el `SeekBar` tras cada movimiento del control arrastable.

Seekbar

Ahora es posible ejecutar la actividad e interactuar con la vista. Pero sin manejar sus eventos asociados dicha vista es bastante inútil. Veamos algunos eventos que nos permitan que se realice algo con el valor del `SeekBar`. En concreto vamos a ver un ejemplo muy sencillo en el que un `TextView` mostrará el valor de un `SeekBar`. Tomemos como ejemplo el siguiente layout:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
                <TextView android:id="@+id/valor"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content" />
                <SeekBar android:id="@+id/seek"
                        android:layout_width="fill_parent"
```

```
                        android:layout_height="wrap_content"
                        android:max="100"/>
</LinearLayout>
```

El evento para el que vamos a crear un manejador es `onProgressChanged`, que se lanzará cuando el usuario arrastre el control del `SeekBar` cambiando su valor. Lo que haremos en dicho manejador es simplemente modificar el valor del `TextView`. Obsérvese que el manejador ha sido creado haciendo que la actividad implemente la interfaz `SeekBar.OnSeekBarChangeListener`:

```
public class EjemploSeekBar extends Activity implements
SeekBar.OnSeekBarChangeListener{

    SeekBar seek;
    TextView valor;

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

        seek = (SeekBar)findViewById(R.id.seek);
                valor = (TextView)findViewById(R.id.valor);
        seek.setOnSeekBarChangeListener(this);
    }

    public void onProgressChanged(SeekBar seekBar, int progress,
        boolean fromTouch) {
                valor.setText(new Integer(progress).toString());
    }

    public void onStartTrackingTouch(SeekBar seekBar) {
                // Hacer algo
        }

        public void onStopTrackingTouch(SeekBar seekBar) {
                // Hacer algo
        }

}
```

El valor del `SeekBar` se ha obtenido en este ejemplo directamente de uno de los parámetros del manejador del evento. Para leer o modificar el valor del `SeekBar` se pueden utilizar también los métodos `setProgress` y `getProgress` que se heredan de la clase `ProgressBar`.

Otros eventos que quizá podrían sernos útiles son `onStartTrackingTouch`, que se lanza en el momento en el que el usuario pulsa sobre el control arrastrable del `SeekBar`, y `onStopTrackingTouch`, que se lanza en el momento en el que se deja de pulsar dicho control.

## 2. Listas

Las listas son un tipo de vista muy importante en Android; son muy utilizadas para mostrar datos al usuario. Una lista muestra un conjunto de elementos ordenados

verticalmente, a través de los cuales se puede navegar por medio de una barra de scroll. Seleccionar un elemento normalmente tendrá como consecuencia otra acción, como por ejemplo lanzar una nueva actividad. En esta sección aprenderemos a crear listas en Android y veremos cómo manejar sus diferentes elementos.

## 2.1. ListActivity

Podemos introducir un `ListView` directamente en un layout como haríamos normalmente con cualquier otro tipo de vista en Android:

```
<ListView
        android:id="@+id/lista"
        android:layout_width="fill_parent"
        android.layout_height="wrap_content">
</ListView>
```

Sin embargo, si el único componente gráfico que va a mostrar nuestra actividad es una lista, es mucho más recomendable hacer que dicha actividad herede de `ListActivity`. Hacer esto simplifica mucho el manejo de listas. Aparte de disponer de todas las funcionalidades de una actividad normal dispondremos de otras relacionadas con listas como por ejemplo un método predefinido para manejar el caso en el que se pulse un elemento de la misma.

Una actividad de tipo `ListActivity` contendrá un adaptador de tipo `ListAdapter` que se encargará de manejar los datos de la lista. Este adaptador deberá ser inicializado en el método `onCreate` de la actividad por medio del método `setListAdapter`. Android proporciona diferentes layouts para asociarlos a los diferentes elementos de un adaptador. Además tendremos la oportunidad de implementar el manejador del evento `onListItemClicked` para realizar alguna acción cuado se escoja una opción de la lista.

## 2.2. Adaptadores

Un `ListView` obtiene los datos a mostrar a través de un adaptador, que debe ser una subclase de `BaseAdapter` y proporciona un modelo de datos que convierta los datos en diferentes elementos de la lista. Android dispone de dos adaptadores estándar: `ArrayAdapter` y `CursorAdapter`. El primero permite manejar datos basados en arrays, como por ejemplo datos almacenados en un `ArrayList`, mientras que el segundo permite manejar datos provenientes de una base de datos. También podemos crear nuestro propio adaptador creando una subclase de `BaseAdapter`.

El método más importante de un adaptador es `getView`, el cual es llamado por cada línea del `ListView` para determinar qué debe ser mostrado en cada fila de la lista y cómo. Normalmente una lista tendrá más elementos de los que caben en la pantalla, por lo que no todos los elementos serán visibles simultáneamente. Es por ello que existe un parámetro llamado `convertView` en `getView` de tal forma que se puedan reutilizar filas y rellenar sus vistas asociadas (cada fila de la lista será una vista) con los nuevos datos a

mostrar.

## 2.3. Nuestra primera ListActivity

Veamos primero cómo crear una `ListActivity` sencilla basada en los elementos que proporciona Android por defecto: utilizaremos un adaptador de tipo `ArrayAdapter` y un layout para las filas predefinido en Android.

En primer lugar creamos un nuevo proyecto que contendrá una única actividad, a la que llamaremos *MiLista*. Esta actividad será una subclase de `ListActivity`. El aspecto que tendrá es el siguiente. Fíjate en que no se llama a `setContentView` desde el método `onCreate` y que por lo tanto no es necesario modificar para nada el archivo de layout *main.xml* que se crea por defecto entre los recursos de la aplicación:

```
public class MiLista extends ListActivity {
        public void onCreate(Bundle icicle) {
                super.onCreate(icicle);
        }
}
```

A continuación vamos a crear un adaptador que sirva para rellenar el listado con los elementos almacenados en un array de cadenas. Debemos añadir lo siguiente al final del método `onCreate`:

```
String[] valores = new String[] {"C", "Java", "C++", "Python", "Perl",
        "PHP", "Haskell", "Eiffel", "Lisp",
        "Pascal", "Cobol", "Prolog"};
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1,valores);
setListAdapter(adaptador);
```

El adaptador es de tipo `ArrayAdapter`. Al constructor le pasamos como parámetro el contexto de la aplicación, el identificador de un layout definido por defecto en Android para mostrar los elementos de la lista (`android.R.layout.simple_list_item_1`) y el array de cadenas. Por último asociamos el adaptador a la lista. Ahora podríamos ejecutar la aplicación y comprobar que efectivamente se muestran los elementos de la lista.

Ejemplo de ListActivity

El siguiente paso podría ser implementar un manjeador para el evento `onListItemClick`. En nuestro caso vamos a hacer que cada vez que se seleccione un elemento de la lista se muestre un `Toast` (ver ejercicios de la primera sesión) mostrando el elemento seleccionado. Añademos el siguiente código dentro de la clase *MiLista*:

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id)
{
        String elemento = (String)getListAdapter().getItem(position);
        Toast.makeText(this, elemento + " seleccionado",
Toast.LENGTH_LONG).show();
}
```

## 2.4. Listas con un layout personalizado

Un siguiente paso lógico podría ser estudiar cómo podemos modificar el layout que se utilizará para visualizar cada elemento de una lista, en lugar de aplicar un layout por defecto, como se hizo en el ejemplo anterior. Ten en cuenta que al crear un layout personalizado éste se asignará a todos y cada uno de los elementos de la lista, por lo que todos ellos se mostrarán con el mismo aspecto. En este ejemplo añadiremos en cada fila una imagen; para ello utilizaremos el archivo *icon.png* que viene incluido por defecto en los recursos de cualquier aplicación Android que creamos desde Eclipse.

Creamos un nuevo fichero llamado *layoutfila.xml* en la carpeta */res/layout/* de nuestra aplicación. Dicho fichero tendrá el siguiente contenido:

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

```
        android:id="@+id/icon"
        android:layout_width="22px"
        android:layout_height="22px"
        android:layout_marginLeft="4px"
        android:layout_marginRight="10px"
        android:layout_marginTop="4px"
        android:src="@drawable/icon">
    </ImageView>

    <TextView
        android:id="@+id/etiqueta"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@+id/etiqueta"
        android:textSize="20px">
    </TextView>
</LinearLayout>
```

El anterior fichero define el layout de cada fila de la lista. Cada una se compondrá de un `LinearLayout` vertical que a su vez contendrá dos elementos: un `ImageView` y el `TextView` donde mostraremos los elementos de la lista. A este `TextView` le hemos dado como valor de su atributo `android:id` el valor `"@+id/etiqueta"`. Será necesario que recordemos esto al inicializar el adaptador.

Modificamos ahora el código de nuestra actividad para que quede de la siguiente forma. El único cambio que estamos realizando es indicar al constructor del adaptador que queremos utilizar nuestro propio fichero *layoutfila.xml* para especificar cómo queremos mostrar los elementos de la lista por pantalla, y que en cada fila la cadena se mostrará en la vista cuyo identificador es `etiqueta` (lo cual se corresponde con el `TextView` en dicho archivo de layout).

```
String[] valores = new String[] {"C", "Java", "C++", "Python", "Perl",
        "PHP", "Haskell", "Eiffel", "Lisp",
        "Pascal", "Cobol", "Prolog"};
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
        R.layout.layoutfila, R.id.etiqueta, valores);
setListAdapter(adaptador);
```


Layout personalizado para los elementos de una lista

## 2.5. El evento onItemLongClick

Otro evento relativo a las listas para el que podemos añadir un manejadir es `onItemLongClick`. Este evento se disparará cuando el usario mantenga pulsado un elemento de la lista durante un periodo de tiempo más prolongado de lo habitual. El manejador se asociará a la lista contenida por el `ListActivity` por medio de una llamada al método `setOnItemLongClickListener`. Esto va a requerir acceder a la vista `ListView` contenida por el `ListActivity`; esto puede hacerse por medio del método `getListView`. A continuación se muestra un código de ejemplo que podría ser añadido al que hemos estado mostrando hasta el momento, al final del método `onCreate`:

```
ListView list = getListView();
list.setOnItemLongClickListener(new OnItemLongClickListener() {
        public boolean onItemLongClick(AdapterView<?> parent, View view,
        int position, long id) {
                Toast.makeText(MiLista.this,
                        "Elemento nº " + position + " pulsado",
                        Toast.LENGTH_LONG).show();
                // Devolvemos el valor falso para evitar que se dispare
                // también el evento onListItemClick
                return true;
        }
});
```

## 2.6. Selección múltiple

Otra de las posibilidades de las listas en Android es permitir que haya varios elementos seleccionados simultáneamente. Para poder indicar el modo de selección de la lista utilizamos el método `setChoiceMode`, pasando como parámetro cualquiera de los dos

valores que se presentan en el siguiente ejemplo:

```
ListView listView = getListView();
// Selección múltiple:
listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
// Selección única:
listView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
```

Para poder acceder a los elementos seleccionados utilizamos o bien el método `getCheckedItemPosition` para el caso de selección única o bien el método `getCheckedItemPositions` para el caso de selección múltiple. Incluso es posible, si se ha establecido un identificador para los elementos, hacer uso del método `getCheckedItemIds`, el cual devolverá el identificador de los elementos seleccionados.

### 2.7. Modificando el layout de un ListActivity

Es posible modificar el layout de una actividad que herede de `ListActivity`. Podríamos definir por ejemplo un layout para el `ListActivity` compuesto por dos elementos `TextView` y una `ListView` entre ambos. Estos `TextView` se podrían utilizar como cabecera o pie de la lista. Si se hace esto hemos de tener en cuenta que se debe asignar al `ListView` el identificador `android:id="@android:id/list"`, ya que éste es precisamente el identificador que busca el `ListView` para visualizar su lista asociada.

## 3. Menús

Los menús son una herramienta estándar en las aplicaciones Android que permite añadir más funcionalidad sin necesidad de sacrificar el valioso y limitado espacio del que disponemos en la pantalla para mostrar elementos. Cada actividad puede tener asociado su propio menú, el cual se mostrará cuando se pulse el botón de menú del dispositivo.

Android también permite la creación de menús contextuales, los cuales pueden ser asociados a una vista cualquiera. El menú contextual de una vista se activa habitualmente cuando ésta vista tiene el foco y se pulsa la pantalla durante al menos tres segundos.

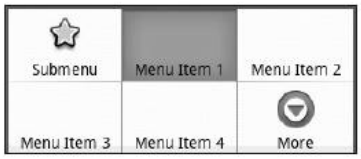
Estos dos tipos de menú soportan la inclusión de submenús, iconos, atajos de teclado y la inclusión de elementos como checkboxes y botones de radio.

### 3.1. El sistema de menús de Android

Android dispone de un sistema de menús para las aplicaciones basado en tres etapas. Este sistema está optimizado para pantallas de pequeño tamaño:

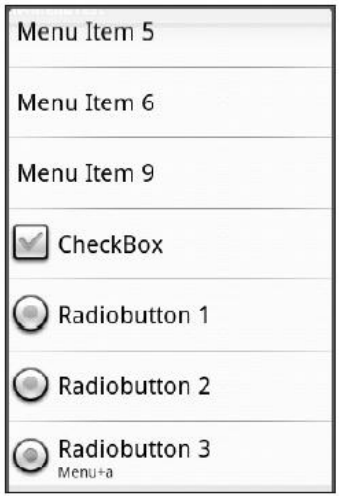
- **Menú de iconos**: se trata de un menú compacto que aparece en pantalla cuando se pulsa el botón correspondiente del dispositivo. El menú muestra los iconos y el texto asociados a un número limitado de opciones (seis como máximo). Normalmente se suelen utilizar iconos en tonos de gris. Este menú no muestra checkboxes, botones de

radio o atajos. Si el menú de la actividad contiene más de seis elementos aparecerá una opción que permitirá mostrar un menú extendido con las opciones adicionales. Pulsando el botón *BACK* en el dispositivo cerramos el menú.




Menú de iconos

- **Menú extendido**: este menú se muestra cuando el menú de iconos tiene demasiadas opciones y se elecciona la opción correspondiente para mostrar los elementos adicionales. Este menú muestra un listado por el que nos podemos mover con una barra de scroll que tan solo contendrá aquellas opciones que no se pudieron mostrar en el menú de iconos. Además, se permite la visualización de atajos, checkboxes y botones de radio. Sin embargo, no se mostrará ningún icono. Pulsando el botón *BACK* en el dispositivo volveremos al menú de iconos.




Menú extendido

- **Submenús**: Android muestra cada submenú en una ventana flotante. Un submenú se visualiza junto con su nombre, y puede contener checkboxes, botones de radio o atajos. Hemos de tener en cuenta que Android no soporta la creación de submenús anidados, por lo que no se podrá añadir un submenú a otro ya existente (intentar esto provocará una excepción). En este caso tampoco se muestran los iconos de las opciones. Al pulsar el botón *BACK* del dispositivo volvemos al menú desde el cual se hizo la llamada al submenú.

## 3.2. Definir el menú de una actividad

Para definir el menú de una actividad se debe sobrecargar su método `onCreateOptionsMenu`. La primera vez que se vaya a mostrar el menú de la actividad se llamará a este método, el cual recibe como parámetro un objeto de la clase `Menu`. A la hora de sobrecargar el método no debemos olvidar llamar al método correspondiente de la superclase, pues éste se encargará de incluir de manera automática opciones adicionales cuando esto sea necesario.

Para añadir opciones a un `Menu` utilizamos el método `add`. Se deben especificar los siguientes datos para cada nuevo elemento de un menú:

- Un entero conocido como identificador de grupo que permita separar los elementos del menú en tareas de procesamiento por lotes u ordenación.
- Un identificador único (un entero) para cada elemento del menú. Esto servirá más adelante en el manejador de eventos adecuado determinar cuál de las opciones de un menú fue seleccionada. Se suele seguir la convención de declarar cada identificador como un atributo privado estático dentro de la actividad correspondiente. Otra opción podría ser utilizar la constante `Menu.FIRST` e ir incrementando este valor en cada elemento posterior.
- Un entero que indique el orden en el que los elementos serán añadidos al menú.
- El texto a mostrar en la opción, ya sea en forma de cadena o ya sea en forma de identificador de recurso (en este caso, el identificador debería corresponderse a una cadena definida en el archivo *strings.xml* de los recursos de la aplicación).

Una vez que terminemos de añadir todas las opciones del menú deberemos devolver `true`. A continuación se muestra un ejemplo en el que se añade una opción al menú de una actividad:

```
static final private int MI_ELEMENTO = Menu.FIRST;

@Override
public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        // Identificador de grupo
        int groupId = 0;
        // Identificador único del evento. Utilizado en
        // el manejador de evento correspondiente
        int menuItemId = MI_ELEMENTO;
        // Posición del elemento en el menú
        int menuItemOrder = Menu.NONE;
        // Texto que mostrará el menú de opciones
        int menuItemText = R.string.menu_item;
        // Creamos el elemento con todos estos datos
        MenuItem menuItem = menu.add(groupId, menuItemId,
                                                menuItemOrder,
menuItemText);
        return true;
}
```

Podemos mantener una referencia al menú creado en `onCreateOptionsMenu` para utilizarla en otras partes del código. Esta referencia será válida hasta que se vuelva a invocar este método. También podemos guardar una referencia a los diferentes objetos `MenuItem` de un menú, aunque también es posible acceder a ellos mediante una llamada

al método `findItem` de la clase `Menu`.

## 3.3. Personalizar elementos de menús

En esta sección resumimos algunas de las opciones de las que disponemos a la hora de diseñar los elementos de nuestros menús.

- **Checkboxes y botones de radio**: estos elementos tan sólo pueden ser visualizados en menús extendidos y en submenús. Para hacer que una opción pase a ser de tipo checkbox utilizamos el método `setCheckable`.Una vez hecho eso podemos controlar su estado mediante el método `setChecked`. Con respecto a los botones de radio, éstos deben ser organizados en grupos. Sólo uno de los botones del grupo de botones de radio podrá estar activo en un momento determinado; al seleccionar cualquiera de ellos, áquel que estuviera seleccionado dejará de estarlo. Para crear un grupo de botones de radio le asignamos el mismo identificador de grupo a todos ellos y a continuación llamamos al método `Menu.setGroupCheckable`, pasando como parámetro dicho identificador de grupo y dándole al parámetro `exclusive` el valor `true`. El siguiente código muestra un ejemplo en el que se añade un elemento de tipo checkbox y un grupo de tres botones de radio:

```
// Creamos un elemento de tipo checkbox
menu.add(0, ELEMENTO_CHECKBOX, Menu.NONE, "CheckBox").setCheckable(true);

// Creamos un grupo de botones de radio
menu.add(GRUPO_BR, BOTONRADIO_1, Menu.NONE, "Opción 1");
menu.add(GRUPO_BR, BOTONRADIO_2, Menu.NONE, "Opción 2");
menu.add(GRUPO_BR, BOTONRADIO_3, Menu.NONE, "Opción 3").setChecked(true);
menu.setGroupCheckable(GRUPO_BR, true, true);
```

- **Atajos de teclado**: es posible asociar un atajo de teclado a un determinado elemento de un menú por medio del método `setShortcut`. Cada llamada a este método requiere en realidad dos teclas, una para poder se usada con el teclado numérico y otra en el caso en el que se esté utilizando un teclado completo. En ninguno de los casos se distinguirá entre mayúsculas y minúsculas:

```
// Añadimos un atajo de teclado a esta opción del menú: '0' en el
// caso de utilizar el teclado numérico, o 'b' en el caso de
// utilizar un teclado completo
menuItem.setShortcut('0','b');
```

- **Texto resumido**: el método `setTitleCondensed` se puede utilizar para especificar el texto asociado a una opción de menú cuando ésta se muestra en el menú de iconos. Teniendo en cuenta que en dicho menú no es posible mostrar checkboxes, a veces este texto se utiliza para indicar el estado de una determinada opción.

```
menuItem.setTitleCondensed("Texto corto");
```

- **Iconos**: una de las propiedades de los elementos del menú es su icono, que se tratará del identificador de un recurso de tipo *Drawable*. Los iconos se muestran únicamente en el menú de iconos, es decir, no se pueden visualizar ni en un menú extendido ni en un submenú. Como se ha comentado anteriormente se suele adoptar la convención de

utilizar imágenes en tonos de gris para los iconos de las opciones de los menús.

```
menuItem.setIcon(R.drawable.icono_opcion);
```

- **Manejador de evento click**: también es posible crear un manejador para el evento de pulsar sobre una opción del menú. Aunque esto sea así, por motivos de eficiencia se desaconseja utilizar esta aproximación; es preferible hacer uso del método `onOptionsItemSelected`, tal como se mostrará una sección posterior.

```
menuItem.setOnMenuItemClickListener(new OnMenuItemClickListener() {
        public boolean onMenuItemClick(MenuItem _menuItem) {
                [ ... hacer algo, devolver true si todo correcto ... ]
                return true;
        }
});
```

- **Intents**: el `Intent` asociado a una opción de menú se activará cuando el evento de seleccionar dicha opción no es manejado ni por `MenuItemClickLister` ni por `onOptionsItemSelected`. Al activarse el `Intent` se hará una llamada a `startActivity` pasando como parámetro dicho `Intent`.

```
menuItem.setIntent(new Intent(this, OtraActividad.class));
```

## 3.4. Actualización dinámica de opciones

Sobrecargar el método `onPrepareOptionsMenu` permite modificar un `Menu` según el estado actual de la aplicación de manera inmediatamente anterior a que dicho menú sea mostrado por pantalla. Esto permite realizar operaciones como añadir o eliminar opciones de manera dinámica, modificar la visibilidad de los diferentes elementos de un menú, o modificar el texto.

Para modificar un determinado elemento de un menú tenemos dos opciones: o bien nos guardamos una referencia al elemento en `onCreateOptionsMenu` cuando éste se crea, o bien hacemos uso del método `findItem` de la clase `Menu`. Este segundo método es el que se utiliza en el siguiente ejemplo:

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
        super.onPrepareOptionsMenu(menu);

        MenuItem menuItem = menu.findItem(ELEMENTO_MENU);
        [ ... modificar el elemento del menú ... ]
        return true;
}
```

## 3.5. Manejo de la selección de elementos

Android maneja toda la actividad relacionada con la selección de opciones de un menú a través de un único manejador de evento: `onOptionsItemSelected`. El elemento del menú que fue seleccionado se pasa a la función como parámetro. Para decidir qué acción realizar, debemos omprar el valor del método `getItemId` del parámetro recibido con

aquellos identificadores que utilizamos a la hora de rellenar el menú con opciones:

```
public boolean onOptionsItemSelected(MenuItem elemento) {
        super.onOptionsItemSelected(elemento);

        // Comprobamos qué elemento del menú fue seleccionado
        switch (elemento.getItemId()) {
                // Comparamos con los identificadores definidos en
                // nuestra actividad
                case (ELEMENTO_MENU):
                [ ... hacer algo ... ]
                return true;
        }

        // Devolvemos false si no hemos hecho nada con el
        // elemento seleccionado
        return false;
}
```

## 3.6. Submenús

Tradicionalmente, en otro tipo de aplicaciones, los submenús se muestran como un árbol jerárquico de opciones. Android presenta un planteamiento alternativo con el objetivo de simplificar la navegación a través de menú en dispositivos cuya pantalla tiene un tamaño reducido. En lugar de una estructura en forma de árbol, un submenú se muestra como una venta flotante.

Para añadir un submenú utilizamos el método `addSubMenu`, el cual recibe los mismos parámetros usados para crear un nuevo elemento de menú mediante el método `add`, lo cual quiere decir que podemos especificar un grupo, un identificador y un texto. También podemos hacer uso de los métodos `setHeaderIcon` y `setIcon` para especificar el icono a mostrar en la cabecera del submenú o en su opción correspondiente en el menú de iconos, respectivamente.

Dentro de un submenú podemos tener cualquier tipo de elementos que también puedan ser añadidos a un menú de iconos o un menú extendido. La única restricción que debemos tener en cuenta es que en Android no es posible tener submenús anidados, es decir, submenús que contengan alguna opción que a su vez se corresponda con un submenú.

El siguiente código muestra un ejemplo en el que dentro de `onCreateMenuOptions` se añade un submenú al menú principal, estableciendo el icono de su cabecera:

```
SubMenu sub = menu.addSubMenu(0, 0, Menu.NONE, "Submenú");
sub.setHeaderIcon(R.drawable.icon);
sub.setIcon(R.drawable.icon);

MenuItem elementoSubmenu = sub.add(0, 0, Menu.NONE, "Elemento submenú");
```

## 3.7. Menús contextuales

Una vista puede tener asociada un menú contextual. Si esa vista tiene el foco y se mantiene pulsada la pantalla durante tres segundos, el menú contextual aparecerá.

Podemos definir menús contextuales y añadirles opciones de la misma forma que se hace con el menú de una actividad.

Para crear un menú contextual sobrecargamos el método `onCreateContextMenu` de la actividad y registramos las vistas de dicha actividad que deben usar el menú por medio del método `registerForContextMenu`. Primero vemos como registrar una vista:

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EditText vista = new EditText(this);
        setContentView(vista);
        registerForContextMenu(vista);
}
```

Una vez que se ha registrado una vista, el método `onCreateContextMenu` será invocado la primera vez que el menú contextual de dicha vista deba ser mostrado. Para que todo funcione correctamente sobrecargamos dicho método, comprobando para qué vista fue lanzado, de tal forma que rellenemos el menú con las opciones correctas.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenu.ContextMenuInfo menuInfo) {

        super.onCreateContextMenu(menu, v, menuInfo);

        menu.setHeaderTitle("Menú contextual");

        menu.add(0, menu.FIRST, Menu.NONE,
                "Elemento 1").setIcon(R.drawable.menu_item);
        menu.add(0, menu.FIRST+1, Menu.NONE,
                "Elemento 2").setCheckable(true);
        menu.add(0, menu.FIRST+2, Menu.NONE,
                "Elemento 3").setShortcut('3','3');
        SubMenu sub = menu.addSubMenu("Submenú");
        sub.add("Elemento de submenú");
}
```

Obsérvese como es posible utilizar con un objeto `ContextMenu` el método `add` de la misma forma que se utiliza con el menú de una actividad, con lo que es posible asociar elementos a un menú contextual como se haría en el caso del menú de una actividad. Esto incluye por supuesto la posibilidad de añadir submenús. Sin embargo, hemos de tener en cuenta que los iconos no se mostrarán.

El evento correspondiente a la selección de un elemento del menú contextual se maneja de la misma forma que en el caso del menú de una actividad. Se puede asociar un `Intent` o un manejador de evento a cada elemento individual, o bien se puede sobrecargar el método `onContextItemSelected` para manejar el evento para el menú en su conjunto, lo cual suele ser una opción más recomendada por cuestiones de eficiencia. El aspecto del manejador sería el siguiente:

```
@Override
public boolean onContextItemSelected(MenuItem item) {

        super.onContextItemSelected(item);
        [ ... hacer algo ... ]
```

```
        return false;
}
```

## 3.8. Definiendo menús como recursos

La alternativa más sencilla a la hora de diseñar el menú de nuestra actividad es por medio de recursos XML. Como en el caso de los layouts, esto nos permite crear menús para diferentes configuraciones de hardware o idiomas. Por ejemplo, una opción podría ser mostrar menús con menos elementos en el caso de pantallas más pequeñas.

Los archivos XML para definir menús se deben almacenar dentro de la carpeta de los recursos de la aplicación, en */res/menu/*. Cada menú se debe crear en un fichero separado. El nombre del fichero pasará a ser el identificador del recurso.

El archivo contendrá un elemento raíz `menu`, que a su vez contendrá varios elementos `item` para especificar cada elemento del menú. Los atributos el elemento `item` son usados para indicar las propiedades del elemento, como su texto asociado, su icono, atajo de teclado, etc. Crear un submenú es tan sencillo como añadir un elemento `menu` en el interior de un elemento `item`.

El siguiente ejemplo muestra cómo definir el menú contextual de ejemplo de la sección anterior como un recurso XML:

```
<menu
                xmlns:android="http://schemas.android.com/apk/res/android"
                android:name="Menú contextual">
        <item
                android:id="@+id/elemento01"
                android:icon="@drawable/elemento_menu"
                android:title="Elemento 1">
        </item>
        <item
                android:id="@+id/elemento02"
                android:checkable="true"
                android:title="Elemento 2">
        </item>
        <item
                android:id="@+id/elemento03"
                android:numericShortcut="3"
                android:alphabeticShortcut="3"
                android:title="Elemento 3">
        </item>
        <item
                android:id="@+id/elemento04"
                android:title="Submenú">
                <menu>
                        <item>
                                android:id="@+id/elemento05"
                                android:title="Elemento de submenú">
                        </item>
                </menu>
        </item>
</menu>
```

Para asociar un menú definido como recurso a una actividad debemos añadir el siguiente código a ésta última:

```
public boolean onCreateOptionsMenu(Menu menu){
        MenuInflater inflater = getMenuInflater();
        // En este caso se usa el fichero de recurso menu.xml
        inflater.inflate(R.menu.menu, menu);
        return true
}
```