Actionbar / Appbar / Toolbar en Android (II)

En el artículo anterior vimos cómo incluir una *action bar* (o *app bar*) en nuestras aplicaciones haciendo uso de la funcionalidad básica incluida por defecto en nuestras actividades al utilizar uno de los temas de la librería de soporte *appcompat* y extender nuestras actividades de `AppCompatActivity`. También vimos cómo personalizar sus características básicas, como colores, título y menús.

Una forma más flexible y personalizable de añadir una action bar a una aplicación es utilizar el nuevo componente `Toolbar` proporcionado por la librería *appcompat*. De esta forma podemos incluir de forma explícita la action bar en nuestros layouts XML como si fuera cualquier otro control, y no sólo en la parte superior de la pantalla a modo de *app bar*, sino también en cualquier otro lugar de la aplicación donde queramos utilizar esta funcionalidad de barra de acciones.

Veremos primero como utilizar el componente `Toolbar` a modo de *action bar*.

Lo primero que debemos asegurar es que nuestro proyecto incluye la última versión de la librería de soporte *appcompat-v7*, en la sección de dependencias del fichero *build.gradle*:

```
1    dependencies {
2        //...
3        compile 'com.android.support:appcompat-v7:22.1.1'
4    }
```

A continuación necesitamos “desactivar” la funcionalidad por defecto que vimos en el artículo anterior configurando un tema para nuestra aplicación que no incluya la action bar. Para ello, editaremos el fichero */res/values/styles.xml*

para hacer que nuestro tema extienda de alguno de los siguientes (dependiendo si queremos partir del tema oscuro o claro):

- `Theme.AppCompat.NoActionBar`
- `Theme.AppCompat.Light.NoActionBar`

En mi caso utilizaré el segundo:

```
<resources>

    <!-- Base application theme. -->
<style name="AppTheme"
parent="Theme.AppCompat.Light.NoActionBar">
<item name="colorPrimary">@color/color_primary</item>
<item
name="colorPrimaryDark">@color/color_primary_dark</item>
<item name="colorAccent">@color/color_accent</item>
</style>

</resources>
```

Como podéis ver podemos definir también los colores principales a utilizar (`colorPrimary`,`colorPrimaryDark` y `colorAccent`), igual que hicimos en el artículo anterior.

Recordemos también que nuestras actividades deben extender a `AppCompatActivity`:

```
1
2
3
4
5
6
```

```
import android.support.v7.app.AppCompatActivity;
//...

public class MainActivity extends AppCompatActivity {
    //...
}
```

Hecho esto, ya podemos modificar el layout de nuestra actividad para incluir la action bar utilizando el nuevo componente `Toolbar`. Veamos cómo quedaría el código y después comentamos los detalles más importantes:

```
<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"
    tools:context=".MainActivity">

<android.support.v7.widget.Toolbar
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:minHeight="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:elevation="4dp"
    android:gravity="bottom"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light" >

        <!-- Resto de la interfaz de usuario -->
```

En mi caso utilizo un `LinearLayout` como contenedor principal, sin márgenes interiores, y en su interior incluyo como primer control el `Toolbar` que actuará como action bar de la actividad. Entre las propiedades asignadas, además de las habituales `id`, `layout_height` y`layout_width`, vemos las siguientes:

- `android:minHeight`. Asignando esta propiedad al alto estandar de una action bar (`?attr/actionBarSize`) conseguimos que los botones de acción y menú de overflow queden siempre en la parte superior de la barra de herramientas aunque incrementemos su tamaño mediante `layout_height` (referencia).
- `android:background`. Asignaremos a esta propiedad el valor `?attr/colorPrimary` de forma que se utilice como color de la barra de herramientas el que hemos definido como `colorPrimary` en el tema de la aplicación (fichero */res/values/styles.xml*).
- `android:elevation`. Esta propiedad define la elevación del componente, lo que determina la sombra que proyectará sobre el elemento inferior. La elevación estadar de la action bar definida en las guías de diseño es de 4dp. Esta propiedad tan sólo tiene efecto cuando la aplicación se ejecuta sobre Android 5.0 o superior, aunque más adelante veremos cómo solucionarlo.

- `android:theme`. Mediante esta propiedad definimos el tema a utilizar por la `Toolbar` (y que heredarán sus controles hijos). No debemos confundir esto con el tema definido a nivel de aplicación en el fichero *styles.xml*. Para conseguir el mismo efecto que en el artículo anterior, donde utilizamos el tema global `Theme.AppCompat.Light.DarkActionBar`(es decir, un tema claro con actionbar oscura) debemos utilizar un tema claro a nivel de aplicación (en nuestro caso vimos antes como utilizamos el tema `Theme.AppCompat.Light.NoActionBar`) y en la `Toolbar` utilizaremos el tema oscuro asignando la propiedad `app:theme`, en este caso con el tema `ThemeOverlay.AppCompat.Dark.ActionBar`.
- `app:popupTheme`. Esta propiedad la utilizaremos sólo si es necesario. En nuestro caso particular lo es, para "arreglar" un efecto colateral de utilizar el tema oscuro para la `Toolbar`. Y es que utilizar este tema también tiene como efecto que el menú de overflow aparezca de color oscuro. Para conseguir que la barra sea oscura pero el menú claro podemos asignar un tema específico sólo a este menú utilizando la propiedad`app:popupTheme`, en nuestro caso el tema `ThemeOverlay.AppCompat.Light`.

Con esto ya tendríamos finalizado el layout XML de la actividad principal con su action bar correspondiente, por supuesto a falta de incluir el resto de controles necesarios para nuestra aplicación (yo incluiré a modo de ejemplo un cuadro de texto y un checkbox, igual que hicimos en el artículo anterior). Por su parte, el menú de overflow se definiría exactamente igual que en el artículo anterior.

Pero nos queda aún un paso importante. En nuestro código debemos indicar que esta `Toolbar` actuará como action bar de la actividad. Para ello, en el método `onCreate()` de la actividad haremos una llamada a `setSupportActionBar()` con la referencia a la toolbar:

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

    //Toolbar Actividad (App bar)
    Toolbar toolbar = (Toolbar) findViewById(R.id.appbar);
    setSupportActionBar(toolbar);

 ......
```

Para no tener que reescribir la definición completa del toolbar en todas nuestras actividades también podemos hacer uso de la cláusula `include`. Para ello, declaramos primero el toolbar en un layout XML independiente, por ejemplo en un fichero llamado */res/layout/toolbar.xml*:

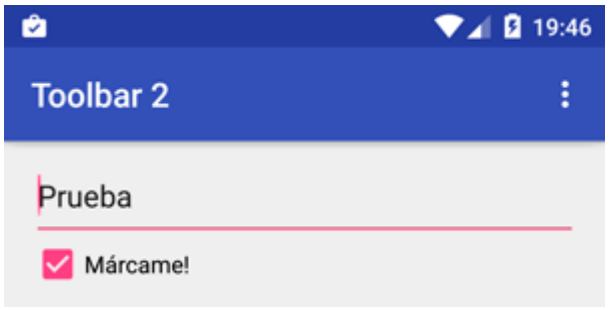
```
1
2       <?xml version="1.0" encoding="utf-8"?>
3       <android.support.v7.widget.Toolbar
4           xmlns:android="http://schemas.android.com/apk/res/android"
5           xmlns:app="http://schemas.android.com/apk/res-auto"
6           android:layout_height="wrap_content"
7           android:layout_width="match_parent"
8           android:minHeight="?attr/actionBarSize"
9           android:background="?attr/colorPrimary"
10          android:elevation="4dp"
11          android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
12          app:popupTheme="@style/ThemeOverlay.AppCompat.Light" >
13
14      </android.support.v7.widget.Toolbar>
15
```

Y posteriormente incluir este fragmento en el layout de nuestras actividades haciendo referencia a él mediante `include`:

```
1
2       <LinearLayout
3           xmlns:android="http://schemas.android.com/apk/res/android"
4           xmlns:tools="http://schemas.android.com/tools"
5           android:layout_width="match_parent"
6           android:layout_height="match_parent"
7           android:orientation="vertical"
8           tools:context=".MainActivity">
9
10          <include android:id="@+id/appbar"
11              layout="@layout/toolbar" />
12
13          <!-- Resto de la interfaz de usuario -->
14
15      </LinearLayout>
16
```

Como podéis ver, en este caso definimos la propiedad `android:id` del control en el `include`, y no en el layout independiente del toolbar.

Ahora sí podríamos ejecutar la aplicación para ver si todo funciona según lo esperado. Si lo hacemos sobre un emulador/dispositivo con Android 5.0 o posterior veremos lo siguiente:



Sin embargo, si lo ejecutamos sobre Android 4.x o anterior, veremos que la actionbar funciona correctamente, pero no se incluye ninguna sombra bajo el control (además de no colorearse la barra de estado, como ya advertimos en el artículo anterior).

Esta sombra se genera debido a la *elevación* definida para el control (propiedad `elevation`), pero esta característica se ha incorporado a partir de Android 5.0 Lollipop y no es compatible con versiones anteriores. Por tanto, si queremos emular el mismo aspecto en versiones de Android anteriores a la 5.0 tendremos que "generar" esta sombra mediante algún método alternativo.

El método que os muestro a continuación está extraido de la aplicación oficial del Google I/O del año 2014, y consiste en añadir al proyecto una imagen (*drawable*) con la sombra y asignarla a la propiedad `foreground` de un `FrameLayout` que "envuelva" al contenido situado bajo la action bar. Se entenderá mejor en el código que veremos a continuación.

En primer lugar añadimos al proyecto la siguiente imagen (*bottom_shadow.9.png*). La colocaremos en la carpeta */res/drawable*. A continuación definiremos un recurso de tipo *drawable* con esta imagen, por ejemplo en un fichero llamado */res/values/refs.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <item name="header_shadow" type="drawable">@drawable/bottom_shadow</item>
</resources>
```

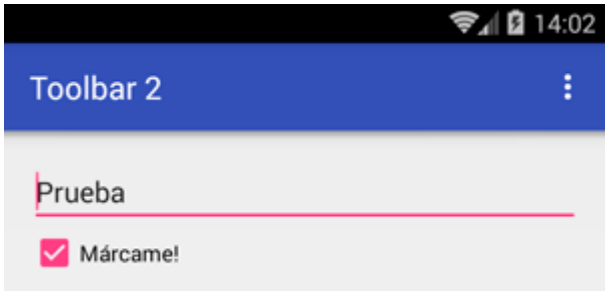
Y por último, tan sólo nos queda utilizar este recurso como foreground de un FrameLayout que envuelva al contenido de nuestra actividad (excluido el toolbar):

```

<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"
    tools:context=".MainActivity">

    <include android:id="@+id/appbar"
        layout="@layout/toolbar" />

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:foreground="@drawable/header_shadow">

        <!-- Resto de la interfaz de la actividad -->

    </FrameLayout>

</LinearLayout>

```

Sin embargo aún queda algo por solucionar. En el caso de que la aplicación se esté ejecutando sobre Android 5.0 o posterior no queremos que se muestre esta imagen, ya que la sombra se genera automáticamente sobre estas versiones. Para ello, lo que podemos hacer es incluir una versión alternativa del fichero *refs.xml* anterior que no utilice la imagen en versiones de Android igual o posterior a Android 5.0 (API 21). Para ello deberemos colocarlo en la carpeta */res/**values-v21***, y al recurso le asignaremos el valor `@null` en vez de la imagen de la sombra:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <item name="header_shadow" type="drawable">@null</item>
</resources>
```

Si ejecutamos ahora la aplicación sobre Android 4.x veremos como la sombra se muestra ya correctamente (en Android 5.x no se apreciará ningún cambio):



Después de todo esto no hemos llegado más que al mismo resultado que en el ejemplo del artículo anterior, pero escribiendo más código. Supongo que te preguntarás, ¿y qué ventajas obtengo? Pues bien, una de las más inmediatas es que de esta forma puedo seguir personalizando la action bar muy fácilmente, por ejemplo modificando su tamaño o añadiendo controles en su interior. Para crear por ejemplo una action bar extendida que además muestre dos líneas de texto con título y subtítulo podríamos modificar de la siguiente forma nuestro fichero toolbar.xml:

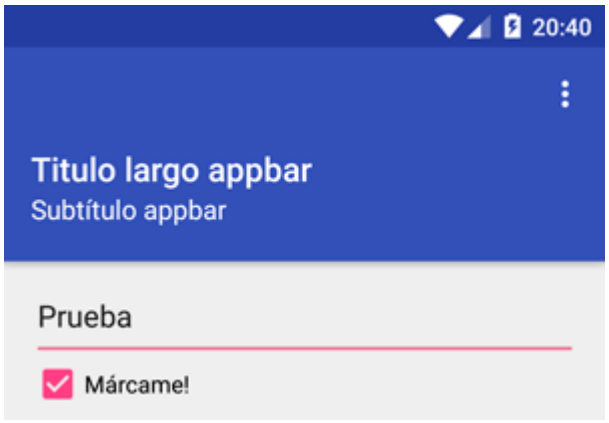
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_height="128dp"
```

```
        android:layout_width="match_parent"
        android:minHeight="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        android:elevation="4dp"
        android:gravity="bottom"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light" >

        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            android:paddingBottom="16dp" >

            <TextView android:id="@+id/txtAbTitulo"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="@string/Título"
                style="@style/TextAppearance.AppCompat.Widget.ActionBar.Title" />

            <TextView android:id="@+id/txtAbSubTitulo"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="@string/Subtítulo"
                style="@style/TextAppearance.AppCompat.Widget.ActionBar.Subtitle" />

        </LinearLayout>

    </android.support.v7.widget.Toolbar>
```

Como podéis ver en el ejemplo anterior, hemos asignado dos nuevas propiedades en el control `Toolbar`. En primer lugar hemos incrementado el alto del control a `128dp`, y además hemos establecido la propiedad `gravity` al valor "`bottom`", de forma que los componentes que añadamos en su interior se alineen sobre el borde inferior.

Adicionalmente, dentro del elemento `Toolbar` hemos añadido dos etiquetas de texto para mostrar el título y el subtítulo, como si se tratara de cualquier otro contenedor. Lo único a destacar es que he utilizado dos estilos predefinidos para estas etiquetas (`AppCompat.Widget.ActionBar.Title` y `AppCompat.Widget.ActionBar.Subtitle`) aunque podría utilizarse cualquier formato para estos elementos.

Con estos cambios nuestra aplicación quedaría así:



Otra ventaja que ya hemos comentado al disponer del control `Toolbar` como componente independiente es que podemos utilizarlo en otros lugares de nuestra interfaz, y no siempre como barra de acciones superior.

Así, podríamos por ejemplo utilizar un componente toolbar dentro de una tarjeta (ver artículo sobre CardView). Para ello, añadamos una tarjeta a nuestra aplicación de ejemplo, y simplemente incluyamos en su interior un control Toolbar de la misma forma que hemos hecho antes:

```
1
2       <android.support.v7.widget.CardView
3           xmlns:app="http://schemas.android.com/apk/res-auto"
4           android:id="@+id/card_view"
5           android:layout_gravity="center"
6           android:layout_width="match_parent"
7           android:layout_height="200dp"
8           app:cardUseCompatPadding="true"
9           app:cardCornerRadius="4dp">
10
11          <android.support.v7.widget.Toolbar
12               android:id="@+id/TbCard"
13               android:layout_height="?attr/actionBarSize"
14               android:layout_width="match_parent"
15               android:minHeight="?attr/actionBarSize"
16               android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
17               app:popupTheme="@style/ThemeOverlay.AppCompat.Light" >
18
19          </android.support.v7.widget.Toolbar>
20
    </android.support.v7.widget.CardView>
```

Si ejecutáramos la aplicación en este momento, la toolbar no se vería ya que no le hemos asignado ningún título ni ningún menú. Anteriormente no tuvimos que hacer esto de forma explícita porque al indicar que la toolbar iba a hacer las función de *app bar* (mediante la llamada a `setSupportActionBar()`), el título y el menú lo tomó automáticamente de la actividad asociada. Sin embargo, en esta ocasión la toolbar es independiente de la actividad, por lo que tendremos que asignar estos elementos nosotros mismos.

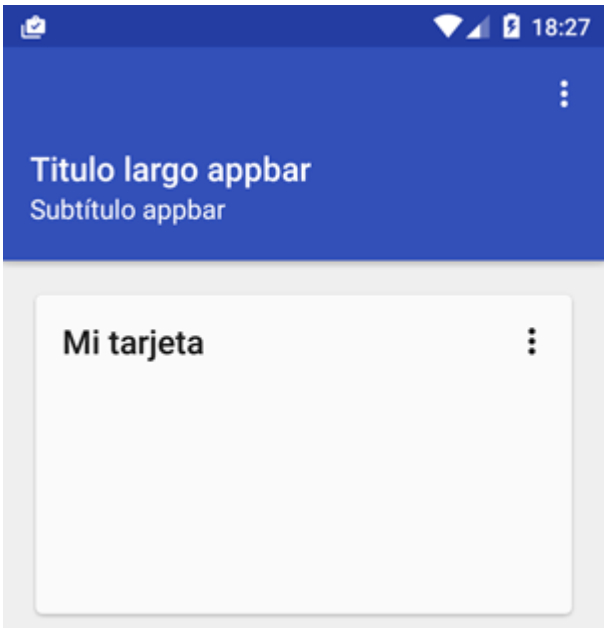
Para ello, desde el método `onCreate()` de la actividad recuperaremos una referencia al control, y llamaremos a sus métodos `setTitle()` e `inflateMenu()` para asignar el título y el menú respectivamente. Para mi caso de ejemplo he definido un nuevo menú menu_tarjeta (definido en el fichero */res/menu/menu_tarjeta.xml*) con dos acciones de muestra:

```
1
2
3     Toolbar tbCard = (Toolbar) findViewById(R.id.TbCard);
4     tbCard.setTitle("Mi tarjeta");
5
6     tbCard.setOnMenuItemClickListener(
7         new Toolbar.OnMenuItemClickListener() {
8             @Override
9             public boolean onMenuItemClick(MenuItem item) {
10
11                  switch (item.getItemId()) {
12                      case R.id.action_1:
13                          Log.i("Toolbar 2", "Acción Tarjeta 1");
14                          break;
15                      case R.id.action_2:
16                          Log.i("Toolbar 2", "Acción Tarjeta 2");
17                          break;
18                      }
19
20                      return true;
21              }
22    });

      tbCard.inflateMenu(R.menu.menu_tarjeta);
```

Vemos también en el código anterior cómo utilizaremos el método `setOnMenuItemClickListener()` para asignar el listener que responderá a las pulsaciones del usuario sobre las opciones del menú. En mi

caso sólo escribo dos mensajes de log, pero obviamente podremos realizar cualquier acción como respuesta.

Si ejecutamos ahora la aplicación de ejemplo debemos ver la nueva tarjeta según lo definido:



Puedes consultar y/o descargar el código completo de los ejemplos desarrollados en este artículo accediendo a la página del curso en GitHub.

Con esto, finalizamos todos los aspectos básicos que quería comentar sobre el nuevo componente Toolbar. En el artículo siguiente veremos dos alternativas de navegación relacionadas con la action bar: filtros (*page filter*) y pestañas (*tabs*).