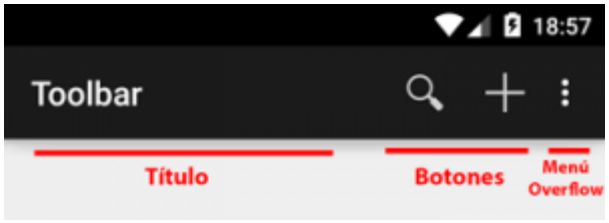
Actionbar / Appbar / Toolbar en Android (I)

La *Action bar*, o *App bar* como se la ha rebautizado con la llegada de Material Design y Android 5.0, es la barra de título y herramientas que aparece en la parte superior de la gran mayoría de aplicaciones actuales de la plataforma Android.

La *app bar* normalmente muestra el título de la aplicación o la actividad en la que nos encontramos, una serie de botones de acción, y un menú desplegable (menú de *overflow*) donde se incluyen más acciones que no tienen espacio para mostrarse como botón o simplemente no se quieren mostrar como tal.



Antes de la llegada de Material Design, a la izquierda del título también podía (y solía) aparecer el icono de la aplicación, aunque esto último ya no se recomienda en las últimas guías de diseño de la plataforma. Lo que sí puede aparecer a la izquierda del título son los iconos indicativos de la existencia de menú lateral deslizante (*navigation drawer*) o el botón de navegación hacia atrás/arriba, pero esto ya lo veremos más adelante.

En la actualidad, Android proporciona este componente a través de la librería de soporte *appcompat-v7*, que podemos incluir en nuestro proyecto añadiendo su referencia en la sección `dependencies` del fichero *build.gradle*:

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

De cualquier forma, en versiones actuales de Android Studio, esta referencia viene incluida por defecto al crear un nuevo proyecto en blanco.

- A partir de aquí, podemos hacer uso de la action bar de dos formas diferentes. La primera de ellas, más sencilla aunque menos flexible, consiste en utilizar la action bar por defecto que se añade a nuestras actividades por tan sólo utilizar un tema (*theme*) determinado en la aplicación y extender nuestras actividades de una clase específica.
- La segunda, más flexible y personalizable aunque requiere más código, consiste en utilizar el nuevo componente `Toolbar` disponible desde la llegada de Android 5.0 (aunque compatible con versiones anteriores). En este primer artículo nos centraremos en la primera de las alternativas indicadas, y en el siguiente hablaremos `Toolbar`.

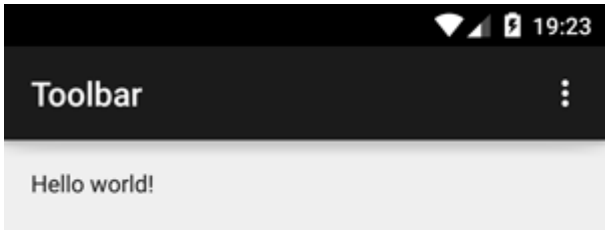
Vamos a comprobar por tanto en primer lugar, aunque también debe venir configurado por defecto con un nuevo proyecto de Android Studio, es que el tema utilizado por nuestra aplicación sea uno de los proporcionados por la librería *appcompat-v7*. Para ello abrimos el fichero *styles.xml* situado en la carpeta*/res/values* y nos aseguramos que el tema de nuestra aplicación extiende de alguno de los temas `Theme.AppCompat` disponibles, en mi caso `Theme.AppCompat.Light.DarkActionBar` (fondo claro con action bar oscura):

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
</style>
```

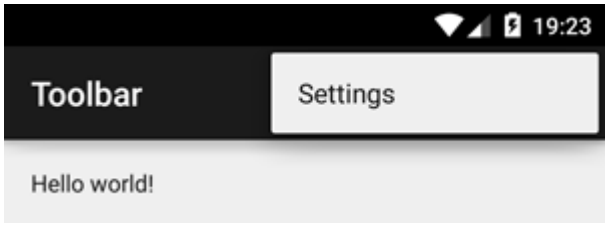
Revisaremos en segundo lugar que nuestras actividades extienden de la clase `AppCompatActivity`, de forma que puedan hacer uso de toda la funcionalidad de la action bar. Hasta la versión 21 de la librería *appcompat-v7* la clase base de la que debían heredar nuestras actividades era `ActionBarActivity`, pero esto cambió con la versión 22, donde la nueva clase a utilizar como base será `AppCompatActivity`. En mi caso, la actividad principal quedaría definida así:

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

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

Tan sólo con esto, si ejecutamos el proyecto podremos comprobar como nuestra actividad muestra la action bar en la parte superior, con el siguiente aspecto:



Si desplegamos el menú de overflow veremos que también aparece una opción por defecto llamada "*Settings*":



Como vemos, la action bar por defecto tan sólo contiene el título y el menú de overflow. El título mostrado por defecto corresponde al título de la actividad, definido en el fichero *AndroidManifest.xml*, en el atributo `label` del elemento `activity` correspondiente a dicha actividad:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
……
<activity
    android:name=".MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
…….
</manifest>
```

Por su parte, los botones de acción y las opciones del menú de overflow se definen de la misma forma que los antiguos menús de aplicación que se utilizaban en versiones de Android 2.x e inferiores. De hecho, es exactamente la misma implementación. Nosotros definiremos un menú, y si la aplicación se ejecuta sobre Android 2.x las acciones se mostrarán como elementos del menú como tal (ya que la action bar no se verá al no ser compatible) y si se ejecuta sobre Android 3.0 o superior aparecerán como acciones de la action bar, ya sea en forma de botón de acción o incluidas en el menú de overflow. En definitiva, una forma de mantener cierta compatibilidad con versiones anteriores de Android, aunque en unas ocasiones se muestre la action bar y en otras no.

¿Y cómo se define un menú de aplicación? Hemos visto como se definen los menús en más detalle pero en este artículo daré las directrices generales para definir uno sin mucha dificultad y utilizar sus opciones..

Un menú se define, como la mayoría de los recursos de Android, mediante un fichero XML, y se colocará en la carpeta */res/menu*. El menú se definirá mediante un elemento raiz `<menu>` y contendrá una serie de elementos `<item>` que representarán cada una de las opciones. Los elementos `<item>` por su parte podrán incluir varios atributos que lo definan, entre los que destacan los siguientes:

- `android:id`. El ID identificativo del elemento, con el que podremos hacer referencia dicha opción.
- `android:title`. El texto que se visualizará para la opción.
- `android:icon`. El icono asociado a la acción.
- `android:showAsAction`. Si se está mostrando una action bar, este atributo indica si la opción de menú se mostrará como botón de acción o como parte del menú de overflow. Puede tomar varios valores:

- `ifRoom`. Se mostrará como botón de acción sólo si hay espacio disponible.
- `withText`. Se mostrará el texto de la opción junto al icono en el caso de que éste se esté mostrando como botón de acción.
- `never`. La opción siempre se mostrará como parte del menú de overflow.
- `always`. La opción siempre se mostrará como botón de acción. Este valor puede provocar que los elementos se solapen si no hay espacio suficiente para ellos.

Al crear un proyecto nuevo en Android Studio, se crea un menú por defecto para la actividad principal, que es el que podemos ver en la imagen anterior con una única opción llamada "*Settings*". Si abrimos este menú (llamado normalmente*/res/menu/menu_main.xml* si no lo hemos cambiado de nombre durante la creación del proyecto) veremos el siguiente código:

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

    <item android:id="@+id/action_settings"
        android:title="@string/action_settings"
        android:orderInCategory="100"
        app:showAsAction="never" />

</menu>
```

Como vemos se define un menú con una única opción, con el texto "*Settings*" y con el atributo `showAsAction="never"` de forma que ésta siempre aparezca en el menú de overflow.

Esta opción por defecto se incluye solo a modo de ejemplo, por lo que podríamos eliminarla sin problemas para incluir las nuestras propias. En mi caso la voy a conservar pero voy a añadir dos más de ejemplo: "Buscar" y "Nuevo", la primera de ellas para que se muestre, si hay espacio, como botón

con su icono correspondiente, y la segunda igual pero además acompañada de su título de acción:

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

    <item android:id="@+id/action_settings"
        android:title="@string/action_settings"
        android:orderInCategory="100"
        app:showAsAction="never" />

    <item android:id="@+id/action_buscar"
        android:title="@string/action_buscar"
        android:icon="@drawable/ic_buscar"
        android:orderInCategory="100"
        app:showAsAction="ifRoom" />

    <item android:id="@+id/action_nuevo"
        android:title="@string/action_nuevo"
        android:icon="@drawable/ic_nuevo"
        android:orderInCategory="100"
        app:showAsAction="ifRoom|withText" />

</menu>
```

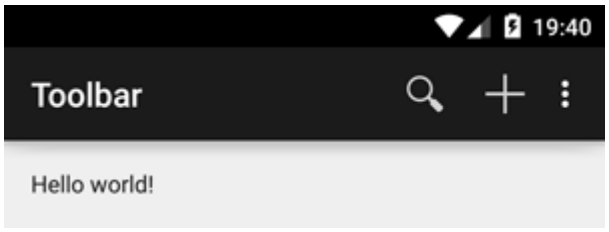
Los iconos `ic_buscar` e `ic_nuevo` los he añadido al proyecto de igual forma que en artículos anteriores, por ejemplo como vimos en el artículo sobre botones.

Como podéis ver además en la segunda opción (`action_nuevo`), se pueden combinar varios valores de `showAsAction` utilizando el caracter "|".

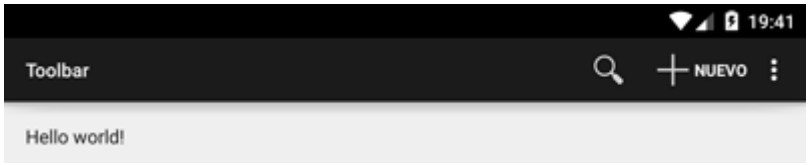
Una vez definido el menú en su fichero XML correspondiente tan sólo queda asociarlo a nuestra actividad principal. Esto se realiza sobrescribiendo el método `OnCreateOptionsMenu()` de la actividad, dentro del cual lo único que tenemos que hacer normalmente es inflar el menú llamando al método `inflate()` pasándole como parámetro el ID del fichero XML donde se ha definido. Este trabajo también suele venir hecho ya al crear un proyecto nuevo desde Android Studio:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is
present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
```

Ejecutemos de nuevo la aplicación a ver qué ocurre:



Como podemos observar, la opción "*Settings*" sigue estando dentro del menú de overflow, y ahora aparecen como botones de acción las dos opciones que hemos marcado como `showAsAction="ifRoom"`, pero para la segunda no aparece el texto. ¿Y por qué? Porque no hay espacio disponible suficiente con la pantalla en vertical. Pero si rotamos el emulador para ver qué ocurre con la pantalla en horizontal (pulsando Ctrl + F12) vemos lo siguiente:



Con la pantalla en horizontal sí se muestra el texto de la segunda opción, tal como habíamos solicitado con el valor `withText` del atributo `showAsAction`.

Podemos seguir personalizando nuestra action bar definiendo los colores principales del tema seleccionado. Esto podemos hacerlo dentro del fichero*/res/values/styles.xml*, y podemos configurar tres colores principales:

- `colorPrimary`. Es el color principal de la aplicación, y se utilizará entre otras cosas como color de fondo de la action bar.
- `colorPrimaryDark`. Es una variante más oscura del color principal, que por ejemplo en Android 5.0 se utilizará como color de la barra de estado (la barra superior que contiene los iconos de notificación, batería, reloj, …). Nota: en Android 4.x e inferiores la barra de estado permanecerá negra.
- `colorAccent`. Suele ser el color utilizado para destacar el botón de acción principal de la aplicación (por ejemplo un botón flotante como el que vimos en el artículo sobre botones, y para otros pequeños detalles de la interfaz, como por ejemplo algunos controles, cuadros de texto o checkboxes.

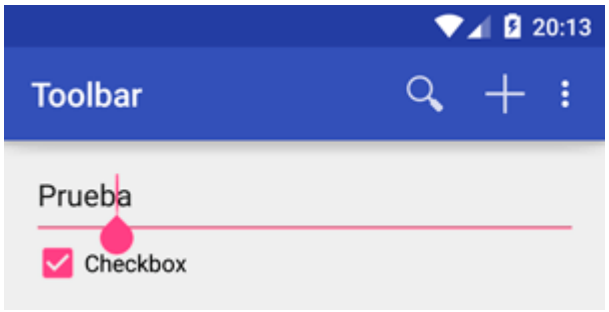
En el siguiente link podéis consultar la paleta de colores estándar definida en las especificaciones de Material Design. En mi caso utilizaré el Indigo intensidad 500 como color principal (`#3F51B5`), la intensidad 700 de esa misma tonalidad como variante más oscura (`#303F9F`), y como color destacado el Pink (`#FF4081`). Los definimos dentro del tema de la aplicación de la siguiente forma:

```
<resources>

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

</resources>
```

La definición de los colores como tal se realiza en un fichero llamado *colors.xml* dentro de la carpeta */res/values* (si no existe el fichero podemos crearlo utilizando el menú contextual sobre la carpeta */res/values*, mediante la opción *New / Values resource file*):

```
<resources>
    <color name="color_primary">#3F51B5</color>
    <color name="color_primary_dark">#303F9F</color>
    <color name="color_accent">#FF4081</color>
</resources>
```

Si ejecutamos de nuevo la aplicación podemos ver los efectos (he añadido a la interfaz algunos controles para ver el efecto sobre ellos del color accent):



Por último, ahora que ya sabemos definir y personalizar los elementos de nuestra action bar queda saber cómo responder a las pulsaciones que haga el usuario sobre ellos. Para esto, al igual que se hace con los menús tradicionales (ver artículo sobre menús para más detalles), sobrescribiremos el método `OnOptionsItemSelected()` de la actividad, donde consultaremos la opción de menú que se ha pulsado mediante el método `getItemId()` de la opción de menú recibida como parámetro y actuaremos en consecuencia. En mi caso de ejemplo tan sólo escribiré un mensaje al log.

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_nuevo:
            Log.i("ActionBar", "Nuevo!");
            return true;
        case R.id.action_buscar:
            Log.i("ActionBar", "Buscar!");;
            return true;
        case R.id.action_settings:
            Log.i("ActionBar", "Settings!");;
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Si ejecutamos ahora la aplicación y miramos el log mientras pulsamos las distintas opciones de la action bar veremos cómo se muestran los mensajes definidos.

En el siguiente artículo veremos cómo incluir la action bar de forma explícita en nuestras aplicaciones (en vez de utilizar la construida por defecto) utilizando el nuevo componente `Toolbar` incluido en las últimas versiones de la librería de soporte.

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