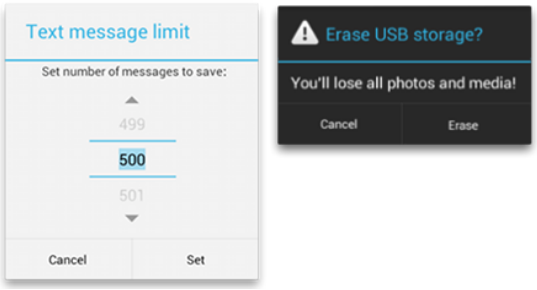
Android Prebe
Clase 3: Alertas y Recursos del Sistema

Cuadros de diálogo

Un diálogo es una ventana pequeña que le indica al usuario que debe tomar una decisión o ingresar información adicional. Un diálogo no ocupa toda la pantalla y generalmente se usa para eventos modales que requieren que los usuarios realicen alguna acción para poder continuar.



Diseño de diálogos

La clase **Dialog** es la clase de base para los diálogos, pero debes evitar crear instancias de **Dialog** directamente. En su lugar, usa una de las siguientes subclases:

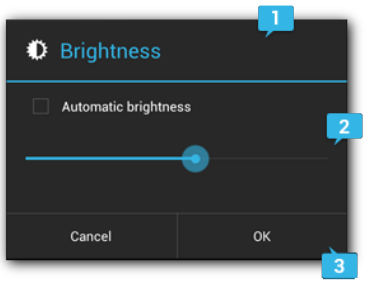
AlertDialog

Un diálogo que puede mostrar un título, hasta tres botones, una lista de elementos seleccionables o un diseño personalizado.

DatePickerDialog o TimePickerDialog

Un diálogo con una IU predefinida que le permite al usuario seleccionar una fecha o una hora.

Estas clases definen el estilo y la estructura para tu diálogo, pero debes usar un **DialogFragment** **como contenedor de tu diálogo**. La clase **DialogFragment** proporciona todos los controles que necesitas para crear tu diálogo y administrar su aspecto, en lugar de llamar a métodos en el objeto **Dialog.**



El uso de **DialogFragment** para administrar el diálogo garantiza que aborde correctamente eventos de ciclo de vida como cuando el usuario presiona el botón *Atrás* o gira la pantalla. La clase **DialogFragment** también te permite reutilizar la IU del diálogo como componente integrable a una IU más grande, al igual que un **Fragment** tradicional (por ejemplo, cuando deseas que la IU del diálogo se vea diferente en pantallas grandes y pequeñas).

Crear un diálogo de alerta

La clase **AlertDialog** te permite crear una variedad de diseños de diálogo y generalmente es la única clase que necesitarás. Como se muestra en la figura 2, hay tres regiones en un diálogo de alerta:

Figura 2: Diseño de un diálogo.

1. **Título**
   Esto es opcional y solo se debe usar cuando el área de contenido esté ocupada por un mensaje detallado, una lista o un diseño personalizado. Si necesitas indicar un mensaje o una pregunta simple (como el diálogo de la figura 1), no necesitas un título.
2. **Área de contenido**
   Esto puede mostrar un mensaje, una lista u otro diseño personalizado.
3. **Botones de acción**
   No debe haber más de tres botones de acción en un diálogo.

La clase **AlertDialog.Builder** proporciona muchas API que te permiten crear un **AlertDialog** con estos tipos de contenido, incluido un diseño personalizado.

Para crear un **AlertDialog**:

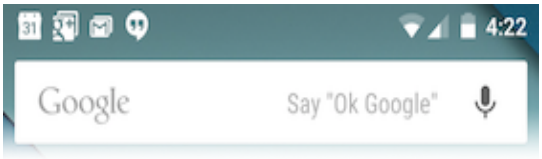
```
// 1. Instantiate an AlertDialog.Builder with its constructor
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
// 2. Chain together various setter methods to set the dialog characteristics
builder.setMessage(R.string.dialog_message)
       .setTitle(R.string.dialog_title);
// 3. Get the AlertDialog from create()
AlertDialog dialog = builder.create();
```

Visitar:https://material.io/guidelines/components/dialogs.html?hl=es-419#dialogs-simple-menus

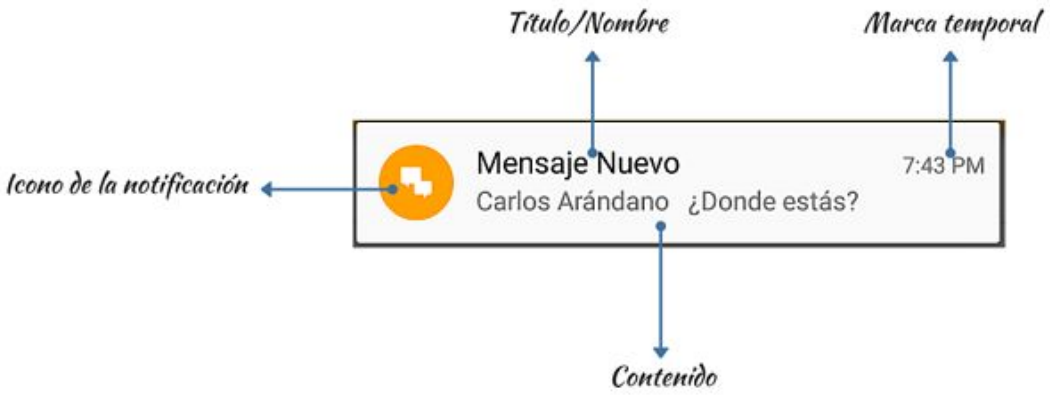
Notificaciones

A notification is a message you display to the user outside of your app's normal UI. When you tell the system to issue a notification, it first appears as an icon in the *notification area*. To see the details of the notification, the user opens the *notification drawer*. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

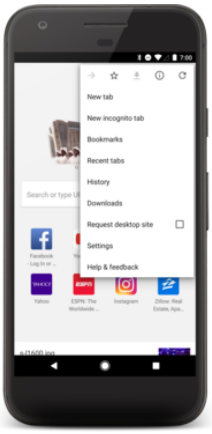
Las notificaciones son ayudas visuales para avisarte sobre eventos importantes que ocurrieron mientras no visualizaban la interfaz de tus aplicaciones



Las notificaciones se visualización por primera vez en el **área de notificaciones en** forma de icono, el cual debe ser lo suficientemente descriptivo para que identifiques intuitivamente lo que está ocurriendo.



Menús



Los menús son un componente común de la interfaz de usuario en muchos tipos de aplicaciones. Para proporcionar una experiencia de usuario conocida y uniforme, debes usar las Menu API para presentar al usuario acciones y otras opciones en las actividades.

Menú de opciones y barra de app

El menú de opciones es la colección principal de elementos de menú para una actividad. Es donde debes colocar las acciones que tienen un impacto global en la app, como “Buscar”, “Redactar correo electrónico” y “Ajustes”.

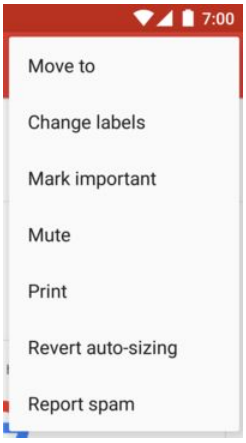
Menú contextual y modo de acción contextual

Un menú contextual es un menú flotante que aparece cuando el usuario hace un clic largo en un elemento. Proporciona acciones que afectan el contenido seleccionado o el marco contextual.

En el menú de acción contextual se muestran los elementos de acción que afectan el contenido seleccionado en una barra en la parte superior de la pantalla y se permite al usuario seleccionar varios elementos.

Menú emergente

Un menú emergente muestra una lista de elementos en una lista vertical que está anclada a la vista que invocó el menú. Es adecuado para proporcionar una ampliación de acciones relacionadas con contenido específico o para proporcionar opciones para una segunda parte de un comando. Las acciones en un menú emergente no deben afectar directamente el contenido correspondiente, para eso están las acciones contextuales. En cambio, el menú emergente es para acciones extendidas relacionadas con partes del contenido de la actividad.



Definición de un menú en XML

Para todos los tipos de menús, Android proporciona un formato XML estándar para definir los elementos del menú. En lugar de incorporar un menú en el código de la actividad, debes definir un menú y todos los elementos en un recurso de menú XML. Luego, puedes agrandar el recurso de menú (cargarlo como un objeto Menú) en la actividad o el fragmento.

El uso del recurso de menú es una práctica recomendada por algunos motivos:

- Es más fácil visualizar la estructura del menú en XML.
- Separa el contenido del menú del código de comportamiento de la aplicación.
- Te permite crear configuraciones alternativas del menú para diferentes versiones de la plataforma, tamaños de pantallas y otras configuraciones aprovechando el framework de recursos de la app.

Para definir el menú, crea un archivo XML dentro del directorio **res/menu/ d**el proyecto y desarrolla el menú con los siguientes elementos:

<menu>

Define un Menú, que es un contenedor para elementos del menú. Un elemento <menu> debe ser el nodo raíz del archivo y puede tener uno o más elementos <item> y <group>.

<item>

Crea un MenuItem, que representa un único elemento en un menú. Este elemento puede contener un elemento <menu> anidado para crear un submenú.

<group>

Contenedor opcional e invisible para elementos <item>. Te permite categorizar los elementos del menú para que compartan propiedades, como el estado de una actividad o visibilidad. Para obtener más información, consulta la sección Creación de grupos del menú.

Ejemplo: game_menu.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/new_game"
        android:icon="@drawable/ic_new_game"
        android:title="@string/new_game"
        android:showAsAction="ifRoom"/>
  <item android:id="@+id/help"
        android:icon="@drawable/ic_help"
        android:title="@string/help" />
</menu>
```

El elemento <item> admite varios atributos que puedes usar para definir la apariencia y el comportamiento de un elemento. Los elementos del menú precedente incluyen estos atributos:

android:id

ID de un recurso que es exclusivo del elemento y permite que la aplicación lo reconozca cuando el usuario lo selecciona.

android:icon

Referencia a un elemento de diseño para usar como el ícono del elemento.

android:title

Referencia a una string para usar como el título del elemento.

android:showAsAction

Especifica cuándo y cómo el elemento debe aparecer como un elemento de acción en la barra de app.

Recursos de diseño

Cómo cambiar la orientación de una aplicación en Android

Para cambiar la orientación de una app en Android, es decir, que los **layouts se muestren como queremos (en horizontal o vertical)** o de ambas maneras, lo que haremos será lo siguiente.
A través del **AndroidManifest.xml podremos barajar el comportamiento de los Activity**.

```
<activity
   android:name="android:name=".MainActivity"
   android:label="@string/song_detail"
   android:screenOrientation="portrait">
</activity>
```

android:screenOrientation="portrait

Solo vertical (evito que se vea en horizontal)

android:screenOrientation="landscape”

Solo horizontal (evito que se vea en vertical)

Pero **si queremos que se muestre de las dos maneras, no tendremos que añadir nada, lo hará automático**.

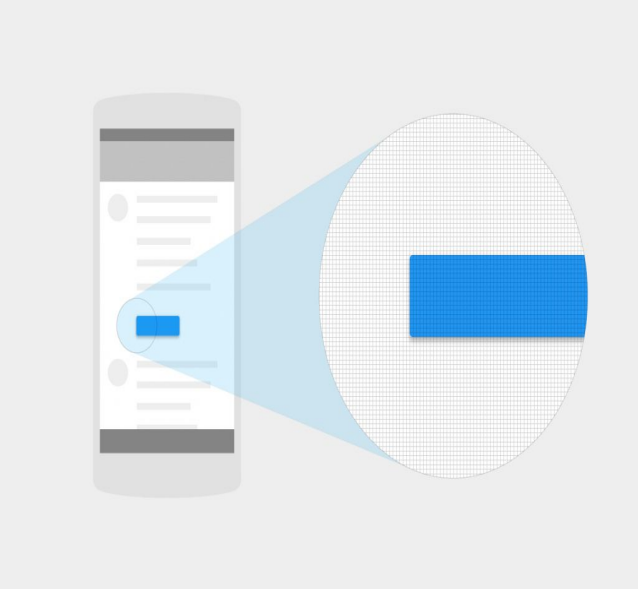
Así que si quieres que te aplicación solo se muestre en vertical, añade "**portrait**", si prefieres que esté girada de siempre "**landscape**", y si te valen ambas, no añadas nada que no hace falta.

Tamaño de pantalla

Unidades y medidas

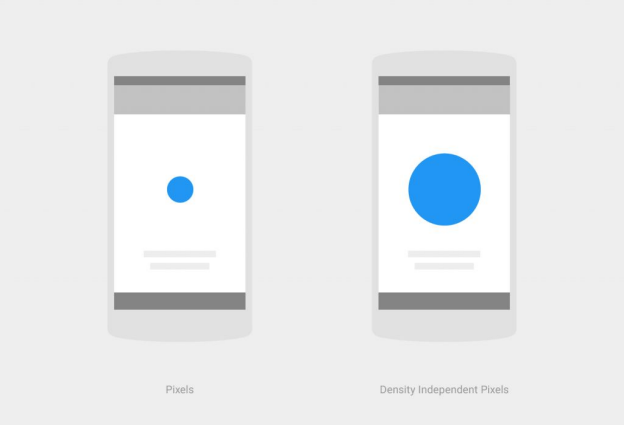
- **Densidad de píxeles**
- **Densidad independiente de píxeles (dp)**
- **Píxeles escalables (sp)**
- **Diseñar Layouts con dp**
- **Escalado de imagen**

Densidad de píxeles



El número de píxeles que encajan en una pulgada se refiere a la "densidad de píxeles". Las pantallas de alta densidad tienen más píxeles por pulgada que los de baja densidad. Como resultado, los elementos de la UI (botones, tarjetas, etc) aparecen más grandes físicamente en pantallas de baja densidad y, más pequeños en las de alta densidad.
DPI, o la resolución de pantalla, se refiere al número de píxeles que hay en una pantalla en particular.
dpi = anchura de la pantalla (o altura) en píxeles/ancho de la pantalla (o altura) en pulgadas.

Densidad independiente de píxeles (dp)



La independencia de densidad se refiere a la muestra uniforme de los elementos de la UI en pantallas con diferentes densidades.
La densidad independiente de píxeles (dpis) es una unidad flexible que escala uniformemente en todas las dimensiones de cualquier pantalla. Cuando desarrolles una aplicación en Android, utiliza *dp* para mostrar elementos de este modo en pantallas de diferentes densidades.
Un dp es igual a un píxel físico en una pantalla de 160 dpi. Para calcular los dp:
dp = (anchura en píxeles * 160) / dpi
Cuando escribas CSS, utiliza px en cualquier lugar donde hayas utilizado dp o sp, ya que dp se utiliza únicamente cuando desarrolles para Android.

Resolución de pantalla	Ancho de pantalla en píxeles (dpi * ancho en pulgadas)	Ancho de pantalla en densidad independiente de píxeles
120 dpi	180 px	
160 dpi	240 px	240 dp
240 dpi	360 px	

Píxeles escalables (sp)

Cuando desarrolles para Android, los píxeles escalables (sp) tienen la misma función que los dp, pero para las fuentes. El valor por defecto de un sp es el mismo que el valor por defecto de un dp.
La principal diferencia entre un sp y un dp es que el sp mantiene los ajustes de la fuente del usuario. Los usuarios que tienen los ajustes personalizados para tener los textos más grandes en accesibilidad verán el tamaño de la fuente relacionado con el tamaño de los textos en las preferencias.

Diseñar Layouts con dp

Cuando diseñes layouts para cualquier pantalla, puedes calcular las medidas de un elemento en dp:
dp = (anchura en píxeles * 160)/dpi
Por ejemplo, un icono de 32 x 32 px en una pantalla de 32′ dpi es igual a 16 x 16 dp.

Escalado de imagen

Las imágenes pueden escalarse hasta verse igual en cualquier resolución de pantalla utilizando los siguientes ratios:

Resolución	dpi	Ratio de píxeles	Tamaño de la imagen (píxeles)
xxxhdpi	640	4.0	400 x 400
xxhdpi	480	3.0	300 x 300
xhdpi	320	2.0	200 x 200
hdpi	240	1.5	150 x 150
mdpi	160	1.0	100 x 100

Prácticas recomendadas

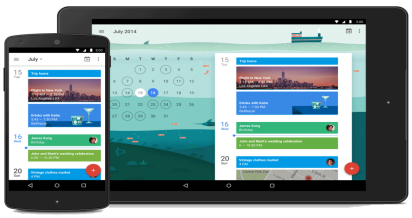
El objetivo de la compatibilidad con varias pantallas es crear una aplicación que pueda funcionar de manera apropiada y verse bien en cualquiera de las configuraciones generalizadas de pantalla compatibles con Android.

1. Usa wrap_content, match_parent o unidades dp al especificar las dimensiones en un archivo de diseño XML.
2. No uses valores de píxeles codificados en el código de tu aplicación.
3. No uses AbsoluteLayout (dejó de estar disponible).
4. Proporciona diferentes elementos de diseño de mapa de bits para diferentes densidades de pantalla.

Soporte de Diferentes Pantallas

Android clasifica las pantallas de los dispositivos en función de dos propiedades generales: el tamaño y la densidad. Deberías preparar tu app para que sea compatible en todos los dispositivos posibles en cualquier rango tanto de tamaño como densidad de pantalla, dando soporte a diferentes pantallas. Por tanto, deberías incluir recursos alternativos que optimicen la apariencia de tu app para dichos pantallas con diferentes tamaños y densidades.

- Hay cuatro tamaños generales: small, normal, large y xlarge.
- Y cuatro densidades generales: low (ldpi), medium (mdpi), high (hdpi), extra high (xhdpi).



También tienes que asegurarte del funcionamiento de tu app en función de la orientación de la pantalla (horizontal o vertical), optimizando la misma para que funcione correctamente en ambos modos y en todas las pantallas posibles.

Soporte de idiomas

Es siempre una buena práctica extraer los strings que aparecen en la UI de tu app y mantenerlos en un archivo externo, string.xml. Android facilita esta tarea con un directorio de recursos para cada proyecto Android. De este modo puedes dar soporte de diferentes idiomas.

Si has creado un proyecto utilizando las Herramientas del SDK Android (visita Crear un Proyecto Android para más información), las herramientas crean un directorio res/ en el nivel superior del proyecto. Dentro de este directorio res/ se encuentran diferentes tipos de recursos en varios subdirectorios. El que nos interesa para esta lección es el res/values/strings.xml, el cual contendrá los valores strings de tu app (las cadenas de texto).

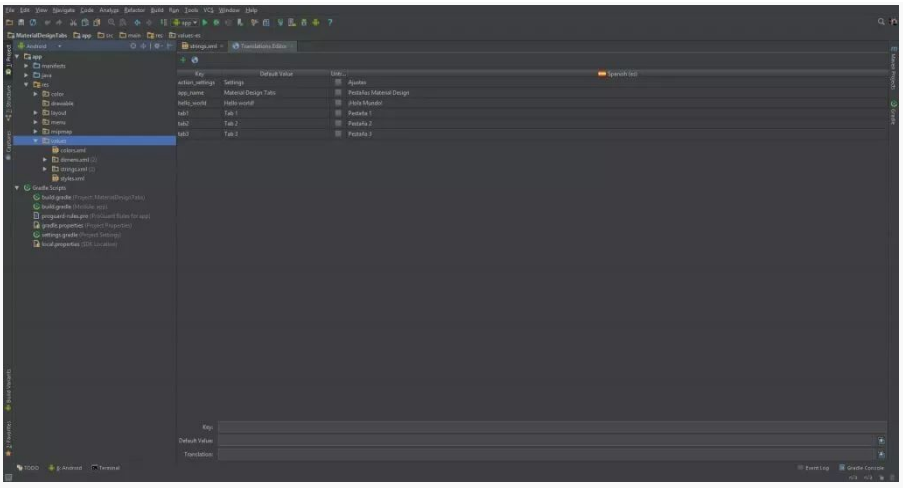


Figura 1. Editor de Traducciones de Android Studio.

Crear Directorios y Archivos de String

Para dar soporte a más idiomas, crea directorios adicionales de values dentro de res/ los cuales incluyen un guión y el código del idioma ISO al final del nombre del directorio. Por ejemplo, values-es/ es el directorio que contiene recursos para el idioma asociado al código “es” (español de España). Android carga los recursos apropiados en tiempo de ejecución en función de los ajustes del dispositivo.Una vez que decidas a qué lenguajes vas a dar soporte, crea los diferentes subdirectorios y los archivos de recursos strings. Por ejemplo:

```
0
1  MyProject/
2      res/
3                                  values/
4                                  strings.xml
5                                  values-es/
6                                  strings.xml
7                                  values-fr/
8                                  strings.xml
9
```

Añade los valores apropiados para cada recurso string dentro de cada uno de dichos archivos strings.xml.
Por ejemplo, aquí tienes diferentes archivos de recursos string para diferentes idiomas.

Inglés, /values/strings.xml:

/values/strings.xml

XHTML

```
0
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string name="title">My Application</string>
4     <string name="hello_world">Hello World!</string>
5 </resources>
6
```

Español de España, /values-es/strings.xml:

/values-es/strings.xml

XHTML

```
0
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string name="title">Mi Aplicación</string>
4     <string name="hello_world">Hola Mundo!</string>
5 </resources>
6
```

French, /values-fr/strings.xml:

/values-fr/strings.xml

XHTML

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mon Application</string>
    <string name="hello_world">Bonjour le monde !</string>
</resources>
```

Nota: Puedes utilizar un calificador local (o cualquier calificador de configuración) sobre cualquier tipo de recurso, por ejemplo, así puedes proporcionar versiones diferentes de imágenes en función del idioma. Para más información, visita Ubicación.

Uso de Recursos String

Puedes hacer referencia a tus recursos string en tu código fuente y otros archivos XML utilizando el nombre del recurso definido por el atributo name del <string>.
En tu código fuente se traduce del siguiente modo: R.string.tu_string. Hay una amplia variedad de métodos que aceptan un recurso de string de este modo.
Por ejemplo:

MainActivity.java

Java

```
// Almacenar el recurso String en una variable String.
String hello = getResources().getString(R.string.hello_world);
// Utilizar directamente el recurso String en un método de un TextView
TextView textView = new TextView(this);
textView.setText(R.string.hello_world);
```

Soporte de Versiones

Aunque que las últimas versiones de Android proporcionan a menudo nuevas APIs para tu app, deberías continuar dando soporte a versiones de Android más antiguas hasta que la mayoría de los dispositivos hayan actualizado.

El dashboard de las Versiones de Plataforma se actualiza regularmente para mostrar la distribución de los dispositivos activos ejecutando cada versión de Android, basándose en el número de dispositivos que visitan la Google Play Store. Generalmente es una buena práctica dar soporte al 90% de los dispositivos activos o más, pero dirigiendo tu app a la última versión.

Consejo: Para proporcionar las mejores características y funcionalidades a lo largo de varias versiones Android, deberías utilizar la Android Support Library en tu app, la cual te permite utilizar las APIs de la plataforma más recientes en las versiones antiguas.

Especificar los Niveles de API Objetivo y Mínimo

El archivo `AndroidManifest.xml` describe los detalles sobre tu app e identidades sobre cuales versiones de Android soporta. Específicamente, el atributo `minSdkVersion` dentro del elemento `<uses-sdk` identifica la API de menor nivel con la que tu app es compatible y la API de mayor nivel para la cual tu app ha sido diseñada y testeada.

AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ... >
    <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="15" />
    ...
</manifest>
```

Las nuevas versiones de Android que son publicadas pueden modificar el comportamiento y los estilos. Permite que tu app tome las ventajas de dichos cambios y asegúrate de que tu app encaja con el estilo de cada dispositivo de los usuarios, por tanto deberías utilizar el valor del atributo `targetSdkVersion` para que sea el de la última versión Android disponible.

Comprobar la Versión del Sistema en Tiempo de Ejecución

Android proporciona un código único para cada versión de plataforma en la clase `Build`. Utiliza dichos códigos dentro de tu app para crear condiciones que aseguren que tu código que depende de los niveles de API superiores es ejecutado únicamente cuando dichas APIs se encuentren disponibles en el sistema.

MainActivity.java

```
private void setUpActionBar() {
    // Make sure we're running on Honeycomb or higher to use ActionBar APIs
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

Tambien de esta forma.

```
0
1 private void setUpActionBar() {
2     // Make sure we're running on Honeycomb or higher to use ActionBar APIs
3     if (Build.VERSION.SDK_INT >= 11 {
4         ActionBar actionBar = getActionBar();
5         actionBar.setDisplayHomeAsUpEnabled(true);
6     }
7 }
8
```

Nota: Cuando analiza recursos XML, Android ignora los atributos XML que no son soportados por el dispositivo actual. Así puedes utilizar atributos XML de forma segura que sean soportados únicamente por versiones superiores sin preocuparte sobre si las versiones antiguas dejarían de funcionar cuando encontrasen dicho código. Por ejemplo, si tu seleccionas targetSdkVersion)"11", tu app incluye la ActionBar por defecto en Android 3.0 o superior. Entonces añadir elementos de menú a la action bar, necesitas utilizar android:showAsAction="ifRoom" en tu recurso de menú XML. Esto es seguro en archivos XML de diferentes versiones, ya que las versiones más antiguas de Android simplemente ignorarán el atributo showAsAction (es decir, no necesitas crear una versión separada en res/menu-v11/).

Uso de Estilos y Temas de la Plataforma

Android proporciona temas para la experiencia de usuario que dan a las apps el aspecto y las sensaciones que transmite el mismo sistema operativo. Esos temas pueden aplicarse a tu app dentro del archivo manifest. Utilizando esos estilos y temas, que ya se encuentran listos para utilizarse, tu app seguirá el último aspecto de forma natural de Android con cada nueva versión publicada.

Para que tu actividad tenga el aspecto de una caja de diálogo:

```
<activity android:theme="@android:style/Theme.Dialog">
```

Para que tu actividad tenga un fondo transparente:

```
<activity android:theme="@android:style/Theme.Translucent">
```

Para aplicar tu propio tema personalizado definido en `res/values/styles.xml`:

```
<activity android:theme="@style/CustomTheme">
```

Para aplicar un tema a tu app al completo (todas las actividades), añade el atributo `android:theme` al elemento `<application>`:

```
<application android:theme="@style/CustomTheme">
```

Referencias:

https://desarrollador-android.com/desarrollo/formacion/empezar-formacion/soporte-de-diferentes-dispositivos/soporte-de-diferentes-idiomas/
https://developer.android.com/guide/topics/ui/notifiers/notifications.html
https://developer.android.com/guide/topics/ui/ui-events.html
https://developer.android.com/guide/practices/screens_support.html?hl=es-419
https://desarrollador-android.com/desarrollo/formacion/empezar-formacion/soporte-de-diferentes-dispositivos/

Leer: https://developer.android.com/about/dashboards/index.html