26.- Estilos y temas

Si tienes experiencia con el diseño de páginas web, habrás visto grandes similitudes entre HTML y el diseño de *layouts*. En los dos casos se utiliza un lenguaje de marcado y se trata de crear diseños independientes del tamaño de la pantalla donde se visualizarán. En el diseño web resultan clave las hojas de estilo en cascada (CSS), que permiten crear un patrón de diseño y aplicarlo a varias páginas. Cuando diseñas los *layouts* de una aplicación, vas a poder utilizar unas herramientas similares conocidas como estilos y temas. Te permitirán crear patrones de estilo que podrán ser utilizados en cualquier parte de la aplicación. Estas herramientas te ahorrarán mucho trabajo y te permitirán conseguir un diseño homogéneo en toda tu aplicación.

Los estilos

Un estilo es una de **colección propiedades que definen el formato y apariencia que tendrá una vista**. Se pueden especificar cosas como tamaño, márgenes, color, fuentes, etc. Un estilo se define en un fichero XML, diferente al fichero XML *Layout* que lo utiliza.

Veamos un ejemplo. El siguiente código:

```
<TextView
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:layout_margin="15dp"
    android:textColor="#00ff00"
    android:typeface="monospace"
    android:gravity="center"
    android:text="¡Hola Mundo!"/>
```

Podemos crear nuestro propio estilo dentro del fichero ***values/styles.xml*** con el siguiente código:

Partiendo de cero:

```
<resources>
    <style name="MiEstilo">
        <item name="android:layout_width">match_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#0D1ADD</item>
        <item name="android:typeface">monospace</item>
        <item name="android:gravity">center</item>
    </style>
</resources>
```

Heredando de un estilo predefinido:

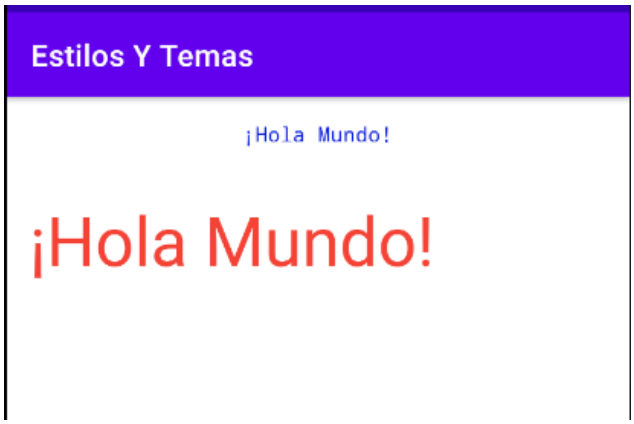
```
<style name="MiEstiloHeredando" parent="TextAppearance.AppCompat.Display1">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#F44336</item>
    <item name="android:layout_marginLeft">15dp</item>
```

```
    <item name="android:textSize">45dp</item>
    <item name="android:gravity">left</item>
</style>
```

Y dentro del Activity:

```
<TextView
    style="@style/MiEstilo"
    android:text="¡Hola Mundo!" />


<TextView
    style="@style/MiEstiloHeredando"
    android:text="¡Hola Mundo!" />
```



Observa como un estilo puede heredar todas las propiedades de un padre (parámetro parent) y a partir de estas propiedades realizar modificaciones.

Heredar de un estilo propio

Si vas a heredar de un estilo definido por ti no es necesario utilizar el atributo `parent`. Por el contrario, puedes utilizar el mismo nombre de un estilo ya creado y completar el nombre con un punto más un sufijo. Por ejemplo:

```
<style name="MiEstilo.grande">
    <item name="android:textSize">25pt</item>
</style>
```

Crearía un nuevo estilo que sería igual a `MiEstilo` más la nueva propiedad indicada. A su vez puedes definir otro estilo a partir de este:

```
<style name="MiEstilo.grande.negrita">
    <item name="android:textStyle">bold|italic</item>
</style>
```

Los temas

Un tema es un **estilo aplicado a toda una actividad o aplicación**, en lugar de a una vista individual. Cada elemento del estilo solo se aplicará a aquellos elementos donde sea posible. Por ejemplo, `CodeFont` solo afectará al texto.

Para aplicar un tema a toda una aplicación edita el fichero *AndroidManifest.xml* y añade el parámetro `android:theme` en la etiqueta `<application>`:

```
<application
    . . .
    android:theme="@style/Theme.EstilosYTemas">
```

También se puede aplicar un tema a una actividad en concreto:

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

Además de crear temas propios se van a poder utilizar algunos disponibles en el sistema. Se puede encontrar una lista de todos los estilos y temas disponibles en Android en: http://developer.android.com/reference/android/R.style.html

Se puede crear un tema propio en `themes.xml`

```
<style name="cursiva" parent="@style/Theme.EstilosYTemas">
    <item name="android:textStyle">italic</item>
    <item name="android:textColor">#FF9800</item>
    <item name="android:textSize">15pt</item>
</style>
```

Ejemplo:

Cambiar toda nuestra aplicación para que tenga fondo negro:

En `themes.xml`:

```
<style name="fondoNegro" parent="@style/Theme.EstilosYTemas">
    <item name="android:textColor">@color/white</item>
    <item
        name="android:windowBackground">@android:color/background_dark
    </item>
</style>
```

En el fichero `AndroidManifest.xml` para la actividad

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

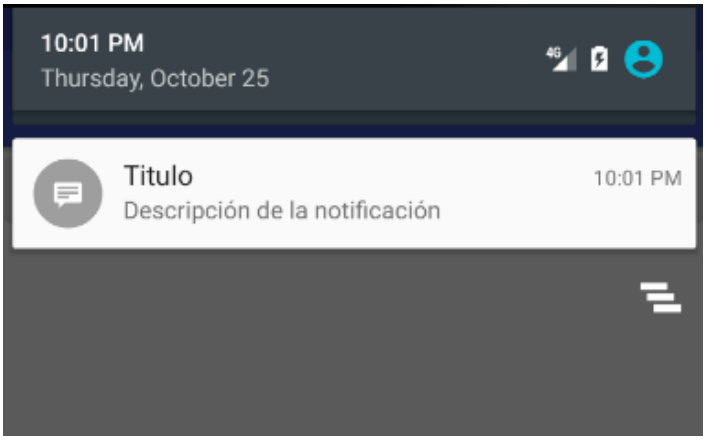
27.- Menús

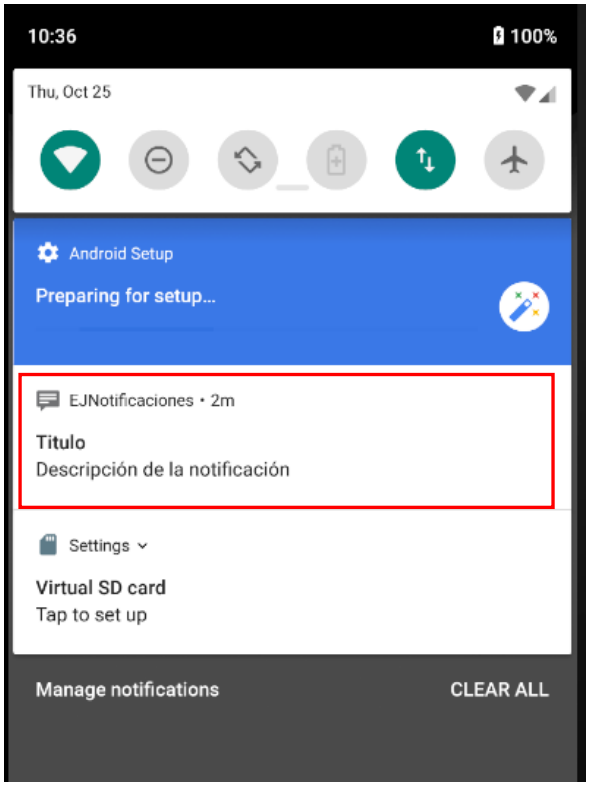
A partir de la versión 3 de Android los menús han caído en desuso debido a la aparición de la *Action bar*. De hecho, si compilas una aplicación con un target igual o superior a la API 11 (Android 3.0) verás cómo los menús que se definen aparecen, no en su lugar habitual en la parte inferior de la pantalla, sino en el menú desplegable de la *Action bar* (que están siendo reemplazadas por las *Toolbar*) en la parte superior derecha.

28.- Notificaciones

La barra de estado de Android se encuentra situadas en la parte superior de la pantalla. La parte izquierda de esta barra está reservada para visualizar notificaciones. Cuando se crea una nueva notificación, aparece un pequeño icono que permanecerá en la barra para recordar al usuario la notificación.

El usuario puede arrastrar la barra de notificaciones hacia abajo, para mostrar el listado de las notificaciones por leer y un mensaje más descriptivo de dichas notificaciones. Un posible ejemplo se muestra a continuación:



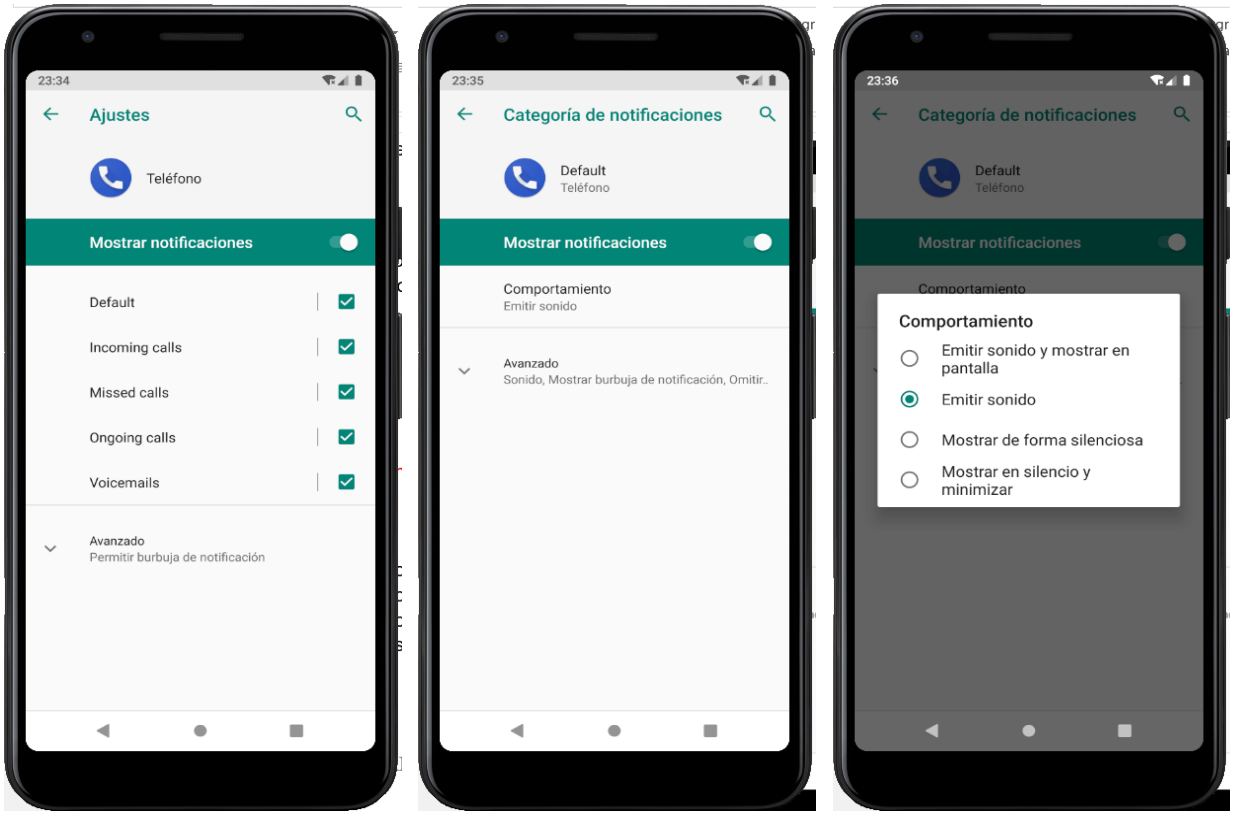


Una notificación puede ser creada por un **servicio** o por una **actividad**. Aunque dado que la actividad dispone de su propio interfaz de usuario, parece que las notificaciones son el mecanismo de

interacción más interesante del que disponen los servicios. Las notificaciones pueden crearse desde un segundo plano, sin interferir con la actividad que en ese momento esté utilizando el usuario.

Para lanzar una notificación desde Android 8 Oreo (API 26) es necesario hacerlo dentro de un canal de notificaión. Si se olvida crear el canal, la notificación no aparecerá. Esto permite a los usuarios tratar todas las notificaciones de un mismo canal de forma conjunta (desactivarla, configurar sonido, …). Una misma aplicación puede crear varios canales, uno por cada tipo de notificaciones. Por ejemplo, en WhatsApp prodríamos crear un canal para cada uno de los grupos.

Para entender mejor este concepto, en un terminal o emulador con versión 8 o superior, accede a *Ajuste → Aplicaciones y notificaciones → Teléfono → Notificaciones de la aplicación*. Se mostrarán todos los canales de notificaciones creados por esta aplicación. Contempla como en Ajustes a los canales se les llama categorías.



Desde esta pantalla se podrá desactivar cualquier canal de notificaciones. Si se pulsa sobre su nombre, se podrá configurar el canal. En Comportamiento se puede configurar la importancia de una notificación para determinar si el usuario ha de ser interrumpido, tanto visual como acústicamente. Los cuatro posibles niveles se muestran a la derecha. También se puede configurar el sonido asociado o si se quiere que aparezcan en modo no molestar.

Crear un nuevo proyecto y asegurarse que se incluye la libería de compatibilidad *appcompat*.

```
dependencies {
    . . .
    implementation 'androidx.appcompat:appcompat:1.1.0'
    . . .
}
```

Un ejemplo sencillo que va a consistir en un único botón que genere una notificación de ejemplo en la barra de estado con la posibilidad de dirigir a una segunda aplicación/actividad que contendrá un `TextView` con un mensaje cuando se pulse sobre ella.

Lo primero que será declarar una serie de variables al inicio de la clase principal `MainActivity`.

```
private NotificationManager notificationManager;
static final String CANAL_ID ="mi_canal";
static final int NOTIFICACION_ID =1;
```

A continuación, en el método `onClick()` del evento de ratón sobre el botón, crearemos la nueva notificación con el siguiente trozo de código:

```
public void onClick(View v) {
      //Creamos notificacion
      notificationManager = (NotificationManager) getSystemService(
                                                  NOTIFICATION_SERVICE);

      //Creamos el canal. SOLO puede hacerse en dispositivos con ver. 8 o más.
      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel notificationChannel = new NotificationChannel(
                        CANAL_ID, "Mis notificaciones",
                        NotificationManager.IMPORTANCE_DEFAULT);

            notificationChannel.setDescription("Descripción del canal");
            notificationManager.createNotificationChannel(notificationChannel);
      }

      NotificationCompat.Builder notificacion =
               new NotificationCompat.Builder(MainActivity.this,CANAL_ID)
                        .setSmallIcon(R.drawable.ic_chats)
                        .setContentTitle("Titulo de la notificación")
                        .setContentText("Este es el texto de la notificacion");

      notificationManager.notify(NOTIFICACION_ID, notificacion.build());

}
```

Analizando este código se ve que lo primero que se necesita es una referencia al `NotificationManager` que permite manejar las notificaciones del sistema.

Lo siguiente será crear el canal. Solo puede hacerse en dispositivos con versión 8 o superior; no obstante, es obligatorio que se cree el canal, de lo contrario, las notificaciones no se ccrearán en dispositivos con estas versiones. Para crear el canal se ha de indicar un *ID*, un *nombre* y un *nivel de importancia*.

De forma predeterminada, todas las notificaciones que se publican en este canal usan el comportamiento visual y auditivo que define el nivel de importancia de la clase `NotifiactionManagerCompat`, como `IMPORTANCE_DEFAULT` y `IMPORTANCE_HIGH`.

La importancia del canal afecta al nivel de interrupción de todas las notificaciones que se publican en este y se debe especificar en el constructor `NotificationChannel`. Se pueden usar uno de los cinco niveles de importancia, que varian desde `IMPORTANCE_NONE` a `IMPORTANCE_HIGH`.

El nivel de importancia que se asigna a un canal se aplica a todos los mensajes de notificaciones que se publican en este.

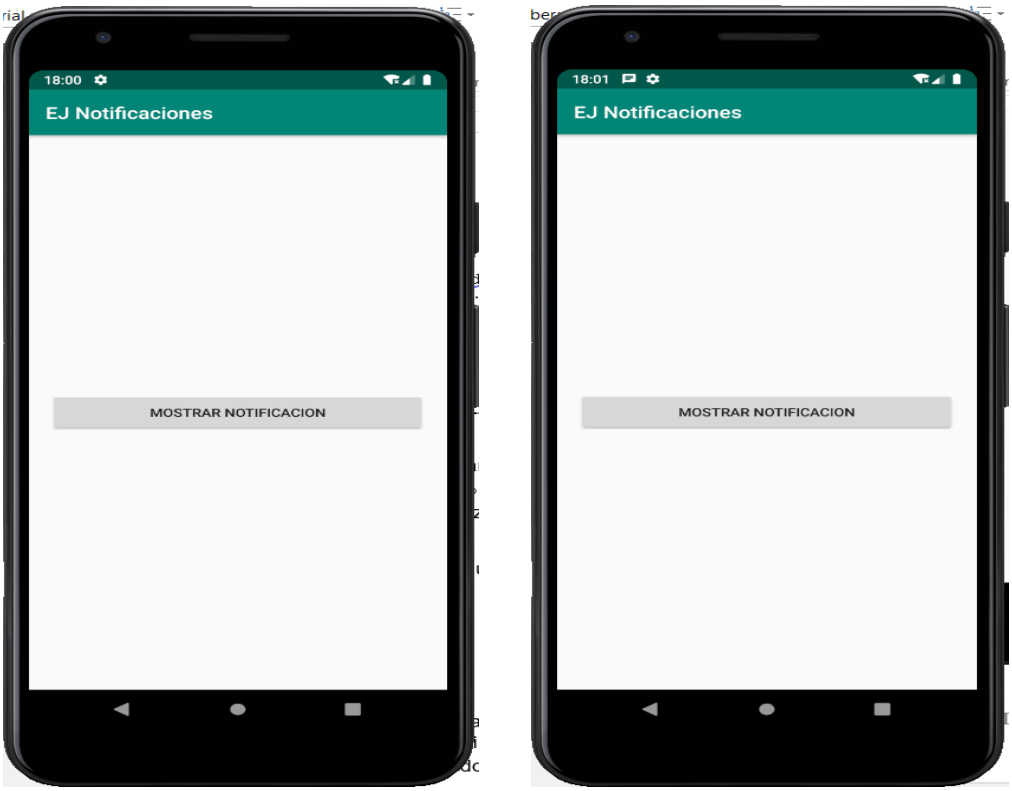
Para agrear compatibilidad con dispositivos que ejecutan Android 7.1 (API 25) o anteriores, también se debe llamar a `setPriority()` para cada notificación por medio de una constante de prioridades de la clase `NotificationCompat`.

NIVEL DE IMPORTANCIA VISIBLE PARA EL USUARIO	IMPORTANCIA (Android 8.0 y posterior)	PRIORIDAD (Android 7.1 y anteriores)
Urgente: Emite un sonido y aparece como una notificación emergente	IMPORTANCE_HIGH	PRIORITY_HIGH PRIORITY_MAX
Alta: Emite un sonido	IMPORTANCE_DEFAULT	PRIORITY_DEFAULT
Media: Sin sonido	IMPORTANCE_LOW	PRIORITY_LOW
Baja: No emite sonido ni aparece en la barra de estado	IMPORTANCE_MIN	PRIORITY_MIN

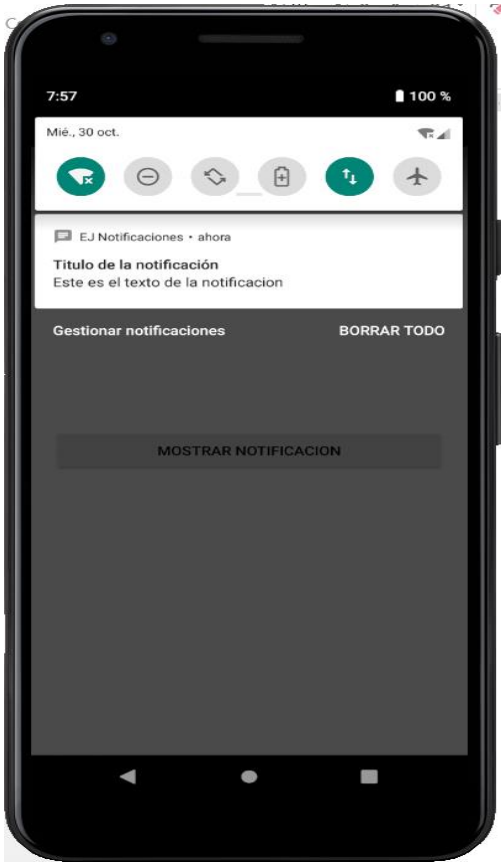
***Tabla 1**: Niveles de importancia del canal*

La notificación se crea utilizando una clase especial `Builder` que dispone de varios métodos set para configurarla. `setContentTitle()` permite indicar *el titulo* que describe la notificaión y `setContextText()` indica *información más detallada*. Con `setSmallIcon()` indicamos el *icono a visualizar*.

El método `notify()` es el encargado de *lanzar la notificación*. En el primer parámetro se indica un *ID* para poder identificar esta notificación en un futuro y en el segundo la notificación.



Cuando arrastrar la barra de notificaciones hacia abajo, para mostrar la lista de notificaciones, al pulsar sobre una de ellas, es habitual que se abra una actividad para realizar acciones relacionadas con la notificación.



Ahora crear una nueva actividad para asociarla a la notificación.

Tras la creación de la varible `notificacion`, añadir el siguiente código:

```
PendingIntent intencionPendiente =
        PendingIntent.getActivity(MainActivity.this, 0,
                new Intent(MainActivity.this, NotificacionActivity.class), 0);
notificacion.setContentIntent(intencionPendiente);
```

Con este código se asocia una actividad que se ejecuará cuando el usuario pulse sobre la notificación. Para ello, se crea un `PendingIntent` (intención pendiente que se ejecutará más adelante) asociado a la actividad `MainActivity`.

También existe la opción de eliminar una notificación desde código, cuando el servicio que la provocó deja de estar activo. Para ello en el método `onDestroy()`, añadimos el siguiente código.

```
notificationManager.cancel(NOTIFICACION_ID);
```

Esto es opcional. Muchas notificaciones han de permanecer visibles aunque el servicio que las creó sea destruido.

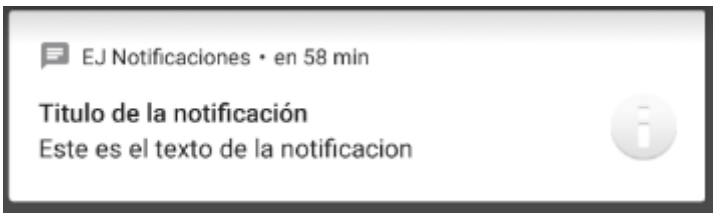
Con lo visto hasta ahora, podemos cubrir la mayoría de las notificaciones, pero estudiaremos como configurar algunas otras características, que pueden ser interesantes.

- `setLargeIcon()` visualiza un icono de mayor resolución que se muestra a la izquerda de la notificación
- `setWhen()` indica la hora en que ocurrio el evento. Solo actúa a efectos de visualización. Aunque se indique una hora futura, la notificación se lanzará inmediatamente

- `setCotentInfo()` muestra un texto ubicado en la parte inferior derecha de la notificación
- `setTicker()` muestra un texto en la barra de estado, durante unos segundos, cuando la notificación se crea por primera vez.

Por ejemplo:

```
notificacion.setLargeIcon(BitmapFactory.decodeResource(getResources(),
                                    android.R.drawable.ic_dialog_info))
        .setWhen(System.currentTimeMillis()+1000*60*60)
        .setContentInfo("más info")
        .setTicker("Texto en la barra de estado");
```



Configurando tipos de avisos en las notificaciones

Una notificación puede utilizar diferentes métodos para alertar al usuario de que se ha producido. Veamos algunas opciones.

Asociar un sonido

Si se considera que una notificación es muy urgente y se desea que el usuario pueda conocerla de forma inmediata, se le puede asociar un sonido que será reproducido cuando se produzca.

El usuario puede definir un sonido por defecto para las notificaciones. Si quieres asociar el sonido de notificaciones por defecto, utiliza la siguiente sentencia en la creación del objeto `NotificationCompat.Builder`:

```
notificacion.setDefaults(Notification.DEFAULT_SOUND);
```

Si prefieres reproducir un sonido personalizado para la notificación, puedes copiar el audio en la carpeta *res/raw* del proyecto, por ejemplo *explosión.mp3*, y añadir la siguiente sentencia:

```
notificacion.setSound(Uri.parse("android.resource://"
                        + getPackageName() + "/" + R.raw.explosion));
```

Añadiendo vibración

También es posible alertar al usuario haciendo vibrar el teléfono. Puedes utilizar la vibración por defecto:

```
notificacion.setDefaults(Notification.DEFAULT_VIBRATE);
```

O por el contrario crear tu propio patrón de vibración:

```
notificacion.setVibrate(new long[] {0,100,200,300});
```

El *array* define un patrón de longitudes expresadas en milisegundos; donde el primer valor es el tiempo sin vibrar, el segundo es el tiempo vibrado, el tercero sin vibrar y así sucesivamente. Este *array* puede ser tan largo como queramos, pero solo será activado una vez, no se repetirá de forma cíclica

Añadiendo parpadeo de LED

Algunos móviles disponen de diodos LED que pueden ser utilizados para avisar al usuario que se ha producido una notificación. Este método es muy interesante si el grado de urgencia del aviso no es lo suficientemente alto como para usar uno de los métodos anteriores. Podemos utilizar el aviso de LED configurado por defecto:

```
notificacion.setDefaults(Notification.DEFAULT_LIGHTS);
```

O podemos definir una cadencia de tiempo y color específica para nuestra notificación:

```
notificacion.setLights(Color.RED, 3000, 1000);
```

En el ejemplo anterior se empieza indicando que queremos que el LED se ilumine en color rojo, durante 3000 ms (3 segundos) y luego esté apagado durante 1000 ms (1 segundo). Esta secuencia se repetirá de forma cíclica hasta que el usuario atienda la notificación.

Conviene destacar que no todos los móviles disponen de un LED para este propósito, y algunos dispositivos con LED no soportan notificaciones con parpadeo. Además, no todos los colores pueden ser utilizados. Otro aspecto a tener en cuenta es que el sistema solo activa el LED cuando la pantalla está apagada.

Actividad propuesta: Notificaciones

Realizar una aplicación para realizar diferentes operaciones matemáticas, todas ellas generadas de forma aleatoria y cuando se llegue a realizar 10 operaciones correctas, lance una notificación.