NOTIFICACIONES AL USUARIO

1. INTRODUCCION

- En ocasiones es necesario mostrar al usuario pequeños mensajes de alerta o de aviso, para los cuales no interesa configurar una pantalla completa.
- Android dispone de diferentes métodos, entre ellos:
  - Los avisos basados en la clase **Toast**.
  - Las **ventanas de diálogo**.
  - Las **notificaciones** en la barra de estado.

2. TOAST

- Aunque aparecen por defecto en la parte inferior de la pantalla, **son personalizables**.
- Su uso defectivo se basa en el código que ya hemos utilizado en multitud de ocasiones:

```
Toast t1 = Toast.makeText(contexto_actual, "Toast por defecto", Toast.LENGTH_SHORT);
t1.show();
// lo que abreviamos mediante
//Toast.makeText(this, "Toast por defecto", Toast.LENGTH_SHORT).show();
```

3. VENTANAS DE DIALOGO

Documentacion en http://developer.android.com/guide/topics/ui/dialogs.html

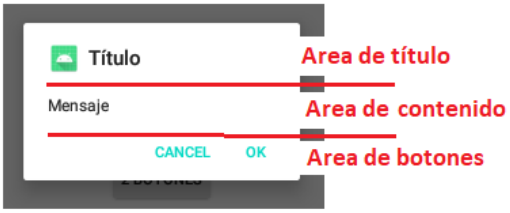
- Un diálogo es una pequeña ventana que aparece delante de la actividad en curso. La actividad que está detrás pierde el foco y el diálogo es el que recibe todas las interacciones con el usuario.
- Se suelen utilizar para que el usuario reciba un mensaje informativo, para pedirle una confirmación rápida o para solicitar que tome una decisión entre varias alternativas.
- No cubren toda la pantalla, sino que flotan sobre la pantalla de fondo, que queda inactiva.
- Android define varios tipos diferentes de ventanas de diálogo:
  - **AlertDialog**: muestra entre cero y tres botones y/o una lista de items seleccionables que pueden incluir casillas de verificación (checkbox) o botones de opción (radio buttons).

- **ProgressDialog**: muestra una barra o rueda de progreso. También se pueden incluir botones (en la documentación se considera obsoleto porque impide que los usuarios interactúen con la aplicación mientras se muestra el progreso)

- **DatePickerDialog**: este diálogo permite seleccionar una fecha.

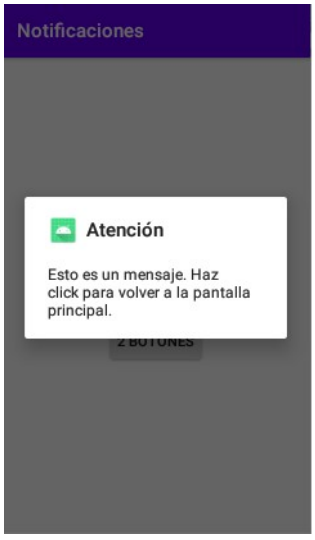
- **TimePickerDialog**: este diálogo permite seleccionar una hora.

- La utilización de las ventanas de diálogo puede basarse en el concepto de "**fragmentos**" (a partir de la API 11 – Android 3.0) pero nosotros, como no tenemos tiempo para ver lo correspondiente a fragmentos, lo veremos sin usar esto.
- Para nosotros, una ventana de diálogo siempre se crea y se muestra como parte de una **Activity**.
- La **forma más simple** de construir una ventana de diálogo es utilizando la subclase ***AlertDialog*** de la clase base ***Dialog***. Esto permite varios diseños que vamos a ver en los siguientes apartados.
- Debemos tener en cuenta que una ventana de diálogo consta de tres zonas:



4. VENTANA CON MENSAJE

- Es una ventana de diálogo que muestra al usuario un mensaje, bloqueando la pantalla hasta que se haga click fuera de la misma. Por ejemplo:



- El proceso a seguir para crear una ventana de diálogo como la anterior es:
  - Crear un objeto ***AlertDialog*** a través de su clase ***Builder***, para la ventana de diálogo.
  - El elemento auxiliar ***Builder*** permite facilitar la creación de los componentes del diálogo a través de sus métodos ***set*():***

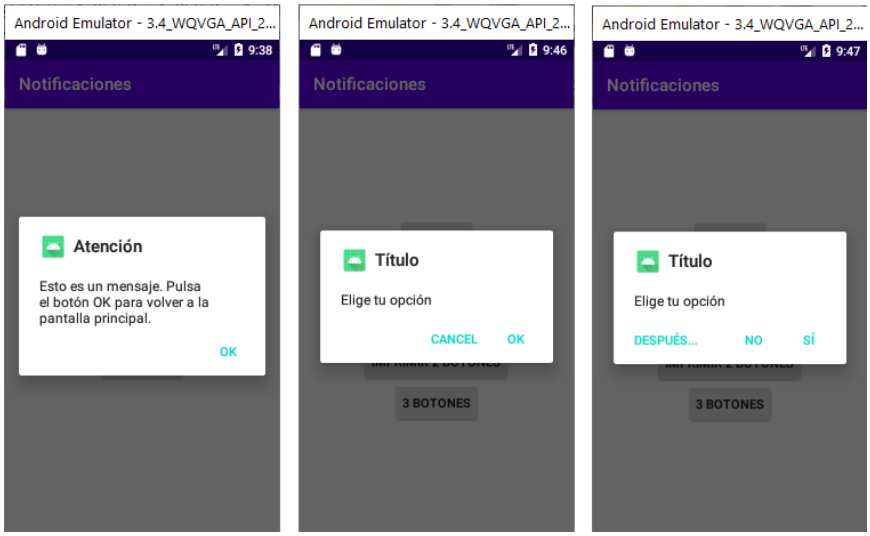
    Configurar el título, el mensaje y el icono de la ventana, con los métodos ***setTitle()***, ***setMessage()*** y ***setIcon()***.

- Por último, pedir que se muestre la ventana mediante el método ***show()***.

```
AlertDialog.Builder ventana = new AlertDialog.Builder(this);
ventana.setTitle("Atención");
ventana.setMessage("Esto es un mensaje. Haz click …");
ventana.setIcon(R.drawable.ic_launcher);
ventana.show();
```

5. VENTANA CON UN BOTON O VARIOS BOTONES

- Puede haber **hasta tres** botones.



- Ventana de diálogo con **un botón**:
  - Igual que en el caso anterior, creamos un objeto ***AlertDialog.Builder***, y configuramos icono, título y mensaje. Pero, antes de pedir que se visualice la ventana, debemos **implementar la existencia del botón**, así como, si queremos, deshabilitar la posibilidad de que la ventana se cierre por otros medios:

```
// Inhabilitamos la posibilidad de que el usuario cierre la ventana sin pulsar el botón
ventana.setCancelable(false);
// Texto a mostrar en el botón y clase anónima que capturará su evento onClick
ventana.setPositiveButton("Ok", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        dialog.cancel();
    }
});
```

  - Lo principal en este tipo de ventanas es la implementación del evento ***onClick*** en forma de objeto **OnClickListener**. Dentro de este evento, hemos cerrado la ventana de diálogo mediante su método ***cancel()***, aunque podríamos haber realizado cualquier otra acción.

- De forma similar procederíamos si la ventana de diálogo tuviese dos o tres botones, empleando los métodos ***setNegativeButton()*** y ***setNeutralButton()***, respectivamente.

6. METODOS ONCREATEDIALOG() Y SHOWDIALOG()

- Aunque hasta ahora, por simplificar, hemos puesto todo el código en el método onCreate() o bien en otros métodos creados por nosotros, **como mejor solución se deberían crear los diálogos dentro del método *onCreateDialog()*** de la Activity a la que están asociados (aunque aparezca como ***deprecated***, su uso es perfectamente válido)

- Cuando se quiere mostrar un diálogo, hay que llamar al método ***showDialog(int)*** y pasarle un integer que identifica unívocamente al diálogo que se quiere mostrar.

```
// Pedimos que se muestre el dialogo asociado a la constante 1
showDialog(DIALOGO_MENSAJE);
```

- Para ello, debemos definir una constante entera para cada ventana de diálogo que queramos crear.

```
// constantes enteras para identificar cada ventana de diálogo
private static final int DIALOGO_MENSAJE = 1;
(…)
```

- Cuando se pide un diálogo por primera vez, Android llama al método ***onCreateDialog(int)*** de la Activity, que es donde se debe instanciar el diálogo. A este método se le pasa el mismo ID que se le había pasado a ***showDialog(int)***. Después de crear el Dialog, se devuelve el objeto al final del método.

```
@Override
protected Dialog onCreateDialog (int id){
    AlertDialog.Builder ventana = new AlertDialog.Builder(this);
    ventana.setTitle("Atención");
    ventana.setMessage("Esto es un mensaje. Pulsa ...");
    ventana.setIcon(android.R.drawable.ic_dialog_alert);
    return ventana.create();
}
```

- El método ***onCreateDialog(int)*** sólo es llamado la primera vez que se demanda el diálogo, después ya queda en la memoria residente y sólo es necesario realizar una llamada con ***showDialog(int)*** para que se muestre en pantalla.

7. VENTANAS DE DIALOGO CON ELEMENTOS DE SELECCION

- Cuando las opciones a seleccionar por el usuario son más de tres podemos utilizar los **diálogos de selección** para mostrar una lista de opciones entre las que el usuario pueda elegir.

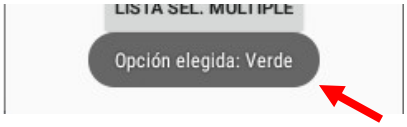
- También utilizaremos la clase ***AlertDialog***, pero sin asignar ningún mensaje, sino que directamente indicaremos la lista de opciones a mostrar, mediante el método ***setItems()*.**

- La lista de opciones puede ser un array tradicional.

- Mediante un listener de tipo ***DialogInterface.OnClickListener*** podemos implementar el evento ***onClick()*** correspondiente a la lista de opciones, para poder saber la opción seleccionada.

- Debido a que la lista aparece en la zona de ventana destinada al mensaje, la ventana de diálogo **no** puede mostrar al mismo tiempo un mensaje y una lista.

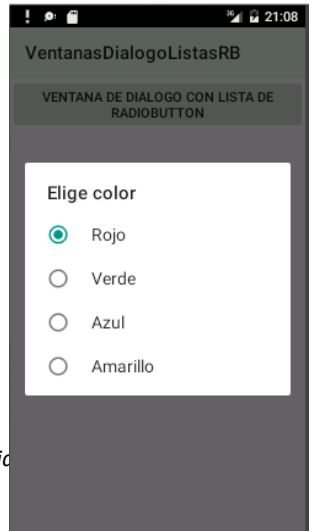
- Ejemplo:

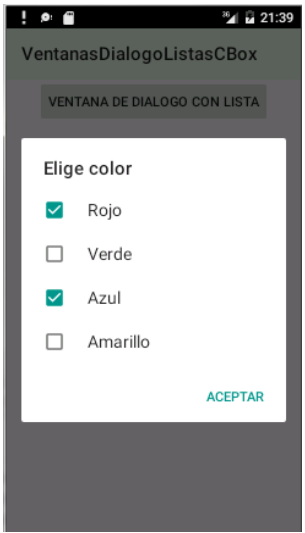


- Código:

```
ventana.setTitle("Elige color");
ventana.setItems(colores, new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        // el parametro "which" contiene la posicion del elemento seleccionado
        Toast.makeText(MainActivity.this, "Opción elegida: " + colores[which],
                Toast.LENGTH_LONG).show();
    }
});
```

- También es posible crear con ventanas de diálogo listas de selección múltiple o listas con botones de radio:





- Para ello, hay que utilizar los métodos ***setMultiChoiceItems()*** (selección múltiple) y ***setSingleChoiceItems()*** (selección simple con botones de radio) **.**

- El método ***setSingleChoiceItems()*** es similar a ***setItems()***, pero recibe como segundo parámetro el índice de la opción marcada por defecto (o el valor -1, si no queremos tener ninguna de ellas marcada inicialmente).

```
ventana.setSingleChoiceItems(colores, 0, new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        (...);
    }
});
```

- El método ***setMultiChoiceItems()*** implementa un listener del tipo **OnMultiChoiceClickListener**. Y puede recibir como segundo parámetro el valor **null** para indicar que no debe aparecer ninguna opción seleccionada por defecto. Además, en este caso, el evento onClick recibe tanto la opción seleccionada (tipo int) como el estado en el que ha quedado dicha opción (tipo boolean).

```
ventana.setMultiChoiceItems(colores, null, new
DialogInterface.OnMultiChoiceClickListener() {
    //ventana.setMultiChoiceItems(colores, checkedItems, new
DialogInterface.OnMultiChoiceClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which, boolean isChecked) {
        if (isChecked) {
            // ...
        } else {
            // ...
        }
    }
});
```

(Ayuda en http://developer.android.com/guide/topics/ui/dialogs.html).

8. NOTIFICACIONES EN LA BARRA DE ESTADO

Documentación en: http://developer.android.com/intl/es/guide/topics/ui/notifiers/notifications.html

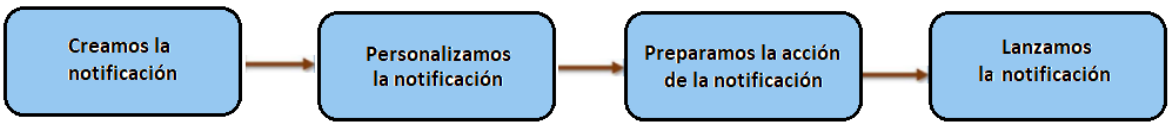
- La barra de estado de Android se encuentra situada en la parte superior de la pantalla. La parte izquierda de esta barra está reservada para visualizar

notificaciones. Cuando se crea una nueva notificación, aparece un pequeño icono que permanecerá en la barra para recordar al usuario la notificación. También, según la versión, puede aparecer inicialmente un texto.

- La visualización puede ser diferente en función de la API utilizada.
- Como usuarios, estamos familiarizados con ellas porque son las que se muestran cuando recibimos un mensaje SMS, hay actualizaciones disponibles, está el reproductor de música funcionando en segundo plano, etc.
- Consisten, básicamente, en un icono (y a veces un texto) que aparece en la barra de estado. Adicionalmente, podemos indicar un mensaje más largo y descriptivo y una marca de fecha/hora que aparece al desplegar la bandeja del sistema.

(Imagen de CABRERA RODRIGUEZ, J: "**Programación multimedia y dispositivos móviles**". Ed. Síntesis)

- Para construir una notificación en la barra de estado hay que seguir el proceso que representa la imagen siguiente:



8.1 CREAR UNA NOTIFICACION

- Hay varias maneras de hacerlo. Una de ellas es mediante la clase ***Notification.Builder***
- Creamos un objeto de dicha clase y le pasamos el contexto de la aplicación:

```
Notification.Builder builder =  new Notification.Builder(this);
```

8.2 PERSONALIZAR LA NOTIFICACION

- Se refiere a **asignar las propiedades** de la notificación mediante sus métodos ***set()***.
- Personalización del área de la barra de estado:

- ***setSmallIcon(int icono)***: Es el icono que aparece en la área de notificación. Debe estar en la carpeta res/drawable, o bien podemos utilizar un drawable del sistema.

- ***setTicker(String mensaje)***: Es el mensaje opcional que aparece durante unos segundos junto al icono pequeño en el área de notificaciones (depende del nivel de API)

```
builder.setSmallIcon(android.R.drawable.star_on);
builder.setTicker("Atención!!");
```

- Personalización de la bandeja del sistema:
  - ***setContentTitle(String)*****:** Título.
  - ***setContentText(String)*****:** Mensaje de la notificación.
  - ***setLargeIcon(Bitmap)***: Icono grande que aparece en la bandeja del sistema.

```
builder.setContentTitle("Aviso!!");
builder.setContentText("Esto es una notificación de aviso");
//forma 1: conversión de un recurso de tipo drawable a un mapa de bits
Bitmap largeIcon=BitmapFactory.decodeResource(getResources(),R.drawable.ic_launcher);
builder.setLargeIcon(largeIcon);
```

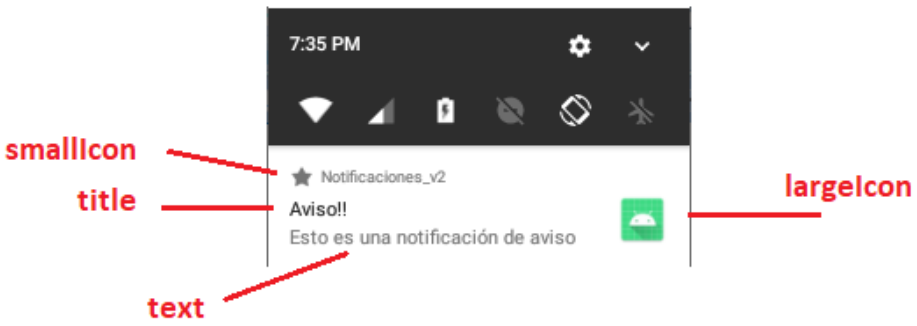
- En este caso, para convertir un recurso de tipo ***drawable*** en un objeto de tipo ***Bitmap*** se ha empleado la clase ***BitmapFactory*** y su método estático ***decodeResource()***

- La clase ***BitmapFactory*** crea objetos Bitmap desde varias fuentes

```
static Bitmap   decodeResource(Resources res, int id)
                Synonym for decodeResource(Resources, int, android.graphics.BitmapFactory.Options) with null Options.
```

- La fecha/hora asociada a la notificación se tomará automáticamente de la fecha/hora actual si no se indica otra cosa.

- Ejemplo de visualización del código anterior con la API 25:



8.3 PREPARAR LA ACCION DE LA NOTIFICACION

- Aunque es opcional, lo normal es que cada notificación tenga asociada una acción. Una acción permite a los usuarios ir directamente desde la notificación a una actividad en la aplicación, donde podrán realizar lo que se considere oportuno. Por

tanto, nos vamos a referir ahora a **cómo establecer la actividad a la cual debemos dirigir al usuario de forma automática si éste pulsa sobre la notificación**.

- Para ello debemos construir un objeto ***PendingIntent***. La clase ***PendingIntent*** tiene una función similar a la clase ***Intent***, excepto que en vez de lanzar la Activity en ese mismo momento, quedará **pendiente hasta un momento posterior**, en nuestro caso, **cuando se pulse sobre la notificación**.

- Para ello **definiremos en primer lugar un objeto *Intent***, indicando la clase de la actividad concreta a lanzar. Este intent lo utilizaremos para construir el PendingIntent final.

- Las instancias de la clase ***PendingIntent*** se crean con el método ***getActivity (Context, int, Intent, int)***.

static PendingIntent	getActivity (Context context, int requestCode, Intent intent, int flags) Retrieve a PendingIntent that will start a new activity, like calling `Context.startActivity(Intent)` .

- Por último, **asociaremos este objeto PendingIntent a la notificación** mediante el método ***setContentIntent()*** de la clase ***Notification.Builder***:

Notification.Builder	setContentIntent (PendingIntent intent) Supply a `PendingIntent` to be sent when the notification is clicked.

```
Intent i =  new Intent(this, ActivityLlamada.class);
PendingIntent pi = PendingIntent.getActivity(this, 0, i, 0);
builder.setContentIntent(pi);
```

8.4 LANZAR LA NOTIFICACION

- Una vez que tenemos nuestra notificación como queremos, es el momento de lanzarla **para que la reciba el usuario**. Esto se hace mediante la clase ***NotificationManager***.

- Laclase ***NotificationManager*** será la encargada de **gestionar** las notificaciones que se deben mostrar en la barra de estado. Por lo tanto, para poder mostrar una notificación necesitaremos crear un objeto de esta clase y llamar a su método ***notify()***, que recibe como segundo parámetro nuestra notificación.

void	notify (int id, Notification notification) Post a notification to be shown in the status bar.

- No se crean instancias de la clase ***NotificationManager*** directamente, sino a través del método ***getSystemService()***.

abstract Object	getSystemService (String name) Return the handle to a system-level service by name.

- El parámetro de tipo String será la constante ***Context.NOTIFICATION_SERVICE.***
- El método ***notify()*** recibe el ID de la notificación (identificador único definido por nosotros. Lo mejor es hacerlo con una constante entera) y un objeto ***Notification*** correspondiente a la notificación que hemos configurado.
- Conseguimos este objeto de clase ***Notification*** llamando al método ***build()*** de la clase ***Notification.Builder***.

Notification	build () Combine all of the options that have been set and return a new `Notification` object.

- Código:

```
NotificationManager nm = (NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);
Notification notificacion=builder.build();
nm.notify(NOTIF_ALERTA_ID, notificacion);
```

(NOTIF_ALERTA_ID es la constante entera que utilizamos como identificador de nuestra notificación).

8.5 CAMBIOS A PARTIR DE LA API 26

- Todo lo anterior es perfectamente válido a nivel conceptual y funcional hasta la **versión 25 (Android 7.1), inclusive**.
- A partir de **la versión 26 (Android 8.0)**, se introduce el concepto de "**canal de notificaciones**", lo que obliga a completar nuestro código anterior para que funcione correctamente.