Programación multimedia y dispositivos móviles

UT-2. Notificaciones al usuario: Toast, Snackbar y AlertDialog

https://developer.android.com/guide/
https://www.sgoliver.net/blog/curso-de-programacion-android/indice-de-contenidos/

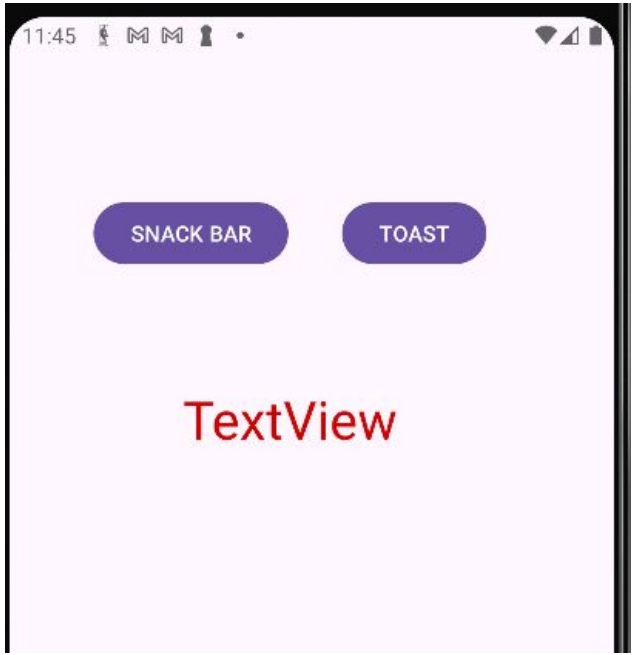
Notificaciones en Android (I): Toast

Un toast es un mensaje que se muestra en pantalla durante unos segundos al usuario para luego volver a desaparecer automáticamente sin requerir ningún tipo de actuación por su parte, y sin recibir el foco en ningún momento (o dicho de otra forma, sin interferir en las acciones que esté realizando el usuario en ese momento). Aunque son personalizables, aparecen por defecto en la parte inferior de la pantalla, sobre un rectángulo gris ligeramente translúcido.

Por sus propias características, este tipo de notificaciones son ideales para mostrar mensajes rápidos y sencillos al usuario, pero por el contrario, al no requerir confirmación por su parte, no deberían utilizarse para hacer notificaciones demasiado importantes.

Introducción

Vamos a ver los tipos de notificación más habituales para el usuario, con y sin iteracción por su parte. Como punto de partida necesitamos un aplicación con 2 botones y un TextView.



Notificaciones en Android (I): Toast

Su utilización es muy sencilla, concentrándose toda la funcionalidad en la clase Toast.

Esta clase dispone de un método estático makeText() al que deberemos pasar como parámetro el contexto de la actividad, el texto a mostrar, y la duración del mensaje, que puede tomar los valores LENGTH_LONG o LENGTH_SHORT, dependiendo del tiempo que queramos que la notificación aparezca en pantalla. Tras obtener una referencia al objeto Toast a través de este método, ya sólo nos quedaría mostrarlo en pantalla mediante el método show().

```
Toast.makeText( context: this,  text: "Hola caracola", Toast.LENGTH_LONG).show()
```

Es relativamente sencillo personalizar un mensaje Toast, pero Google no permite hacer uso de las personalizaciones a partir del API 30 así que no vamos a verlas.

Notificaciones en Android (II): Snackbar

A raíz de la aparición de Android 5 Lollipop, Google introdujo un nuevo tipo de notificación: son los llamados **snackbar**.

Un **snackbar** debe utilizarse para mostrar feedback sobre alguna operación realizada por el usuario, y es similar a un **toast** en el sentido de que aparece en pantalla por un corto periodo de tiempo y después desaparece automáticamente, aunque también presenta algunas diferencias importantes, como por ejemplo que puede contener un botón de texto de acción.

Un ejemplo típico de snackbar con acción es cuando eliminas un correo del Gmail donde te avisa y te da la opción de deshacer.

Notificaciones en Android (II): Snackbar

Según las especificaciones del componente dentro de Material Design, un **snackbar** debe mostrar mensajes cortos, debe aparecer desde la parte inferior de la pantalla, no debe mostrarse más de uno al mismo tiempo, puede contener un botón de texto para realizar una acción, y normalmente puede descartarse deslizándolo hacia un lateral de la pantalla (aunque para esto último necesitaremos hacer algún ajuste).

Para mostrar un **snackbar** procederemos de una forma muy similar a cómo lo hacíamos en el caso de los toast. Construiremos el snackbar mediante el método estático **Snackbar.make()** y posteriormente lo mostraremos llamando al método **show()**.

Vamos a hacerlo desde el botón de Snackbar que tenemos en la app.

Notificaciones en Android (II): Snackbar

```
Snackbar.make(myConstraint,  text: "Esto es una prueba", Snackbar.LENGTH_LONG).show()
```

En este caso el método make() recibe 3 parámetros.

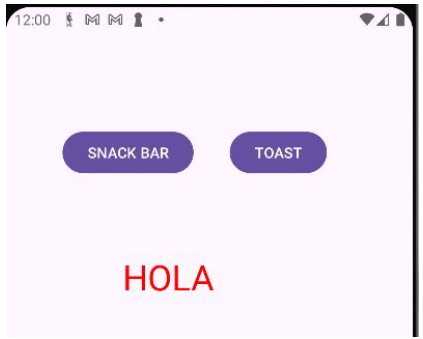
El segundo y el tercero son equivalentes a los ya mostrados para los toast, es decir, el texto a mostrar en el snackbar y la duración del mensaje en pantalla (que en este caso podrá ser Snackbar.LENGTH_SHORT o Snackbar.LENGTH_LONG).

El primero requiere más explicación. Como primer parámetro debemos pasar la referencia a una vista que permita al snackbar (navegando hacia arriba por la jerarquía de vistas de nuestro layout) descubrir un «contenedor adecuado» donde alojarse. Este contenedor será normalmente el content view o vista raíz de la actividad. En mi caso, MyContraint es un “casteo” del ConstraintLayout “main” de la actividad.

Notificaciones en Android (II): Snackbar

Si queremos añadir un botón, y la lógica asociada, se implementa con el método setAction. A este método le pasaremos como parámetros el texto del botón de acción, y un listener para su evento onClick donde podremos responder a la pulsación del botón realizando las acciones oportunas (en mi caso pintando en el textview un mensaje):

```
Snackbar.make(myConstraint,  text: "Esto es una prueba", Snackbar.LENGTH_LONG)
    .setAction( text: "Acción", View.OnClickListener {
        myTextView.text = "HOLA"
        myTextView.setTextColor(Color.RED)
        myTextView.setTextSize(34.0F)
    })
    .show()
```



Notificaciones en Android (II): Snackbar

Podemos especificar el color de la letra de la acción en la construcción del snackbar mediante el método setActionTextColor(), pasándole directamente un color, o recuperando alguno de los definidos en los recursos de nuestra aplicación mediante getResources().getColor(id_recurso_color,null)

```
Snackbar.make(myConstraint,  text: "Esto es una prueba", Snackbar.LENGTH_LONG)
    .setActionTextColor(getResources().getColor(R.color.miVerde,  theme: null))
    .setAction( text: "Acción", View.OnClickListener {
        myTextView.text = "HOLA"
        myTextView.setTextColor(Color.RED)
        myTextView.setTextSize(34.0F)
    })
    .show()
```

activity_main.xml | MainActivity.kt | colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="miVerde">#39FF14</color>
</resources>
```

Notificaciones en Android (II): Snackbar

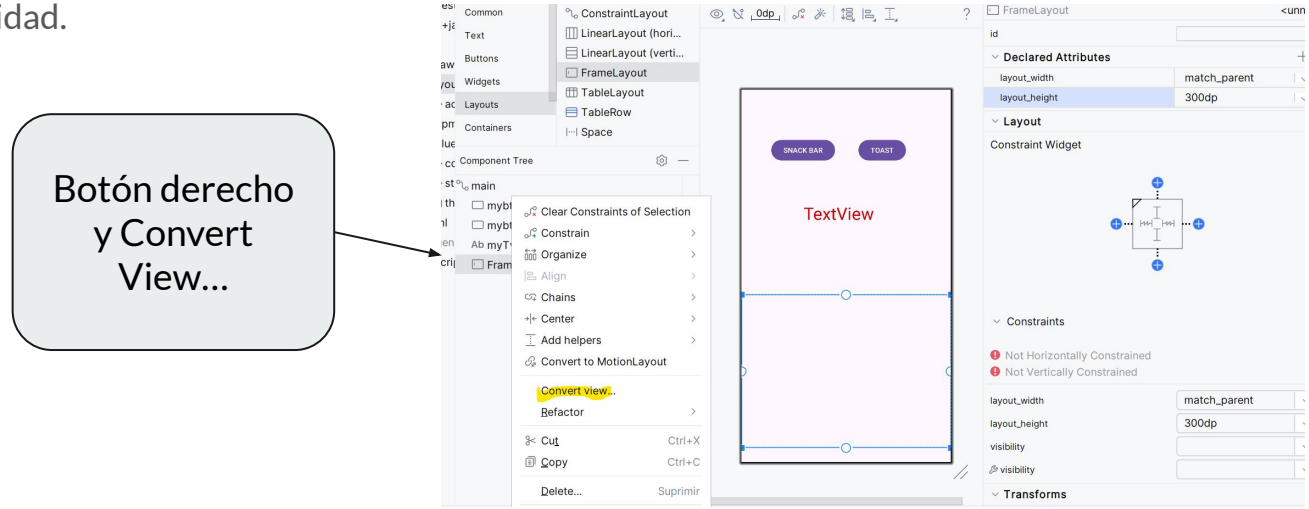
Por último, vamos a ver la posibilidad de descartar un snackbar con el gesto de deslizamiento hacia la derecha. Si intentamos hacer esto sobre el snackbar que tenemos ahora veremos que no es posible.

Para poder hacerlo el snackbar debe generarse dentro de un tipo de contenedor llamado **CoordinatorLayout**. Este nuevo componente que a simple vista parece un FrameLayout puede utilizarse para gestionar de una forma semiautomática algunas de las animaciones habituales de una aplicación android actual. Veremos si dedicamos más tiempo a este componente, pero por el momento vamos a conformarnos con saber que si añadimos como elemento “padre” de nuestro snackbar un layout un CoordinatorLayout, nuestros snackbar podrán automáticamente descartarse deslizándolos a la derecha.

Vamos a hacer esto de 2 formas distintas.

Notificaciones en Android (II): Snackbar

Añadimos un Layout de cualquier tipo a nuestra Activity. Yo he cogido un FrameLayout, pero da igual cual cojas. Lo hacemos match_parent de ancho, 300 dp de alto y lo colocamos en la parte de abajo de la actividad.



Notificaciones en Android (II): Snackbar

Lo “anclamos” al constraint, le ponemos el id y casteamos el id en la actividad. Utilizamos este objeto como el view que contiene el snackbar

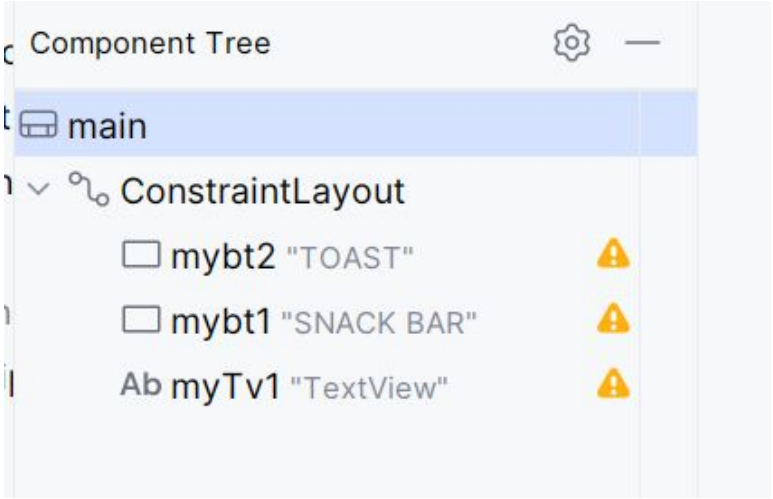
```
lateinit var myCoordinator: CoordinatorLayout

Snackbar.make(myCoordinator, text: "Esto es una prueba", Snackbar.LENGTH_LONG)
    .setActionTextColor(getResources().getColor(R.color.miVerde, theme: null))
    .setAction( text: "Acción", View.OnClickListener {
        myTextView.text = "HOLA"
        myTextView.setTextColor(Color.RED)
        myTextView.setTextSize(34.0F)
    })
    .show()
```

Notificaciones en Android (II): Snackbar

Esto tiene el inconveniente de que ponemos un layout en una zona de la actividad que si hay otros controles hay que encajarlo (puedes meter controles en el coordinator que a nivel de colocación funciona como un FrameLayout).

Otra alternativa es hacer a tu Layout principal de tipo coordinator layout, poner dentro un ConstraintLayout y colocar dentro ya los controles como quieras.



Notificaciones en Android (III): AlertDialog

Otro mecanismo que podemos utilizar para mostrar o solicitar información puntual al usuario son los cuadros de diálogo.

En principio, los diálogos de Android los podremos utilizar para:

- ❏ Mostrar un mensaje solicitando confirmación al usuario de que lo ha leído.
- ❏ Solicitar al usuario una elección (simple o múltiple) entre varias alternativas.

Básicamente un AlertDialog es una ventana superpuesta a toda la aplicación en la que se muestra un mensaje, oscureciendo ligeramente el fondo para centrar la atención del usuario.

Al igual que pasaba con los mensajes Toast y Snackbar, la implementación del AlertDialog que viene "de caja" con Android es bastante sencillo. De cualquier forma, veremos también cómo personalizar completamente un diálogo para adaptarlo a cualquier otra necesidad.

Notificaciones en Android (III): AlertDialog

AlertDialog - Un botón de confirmación

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.

AlertDialog - Un botón de confirmación

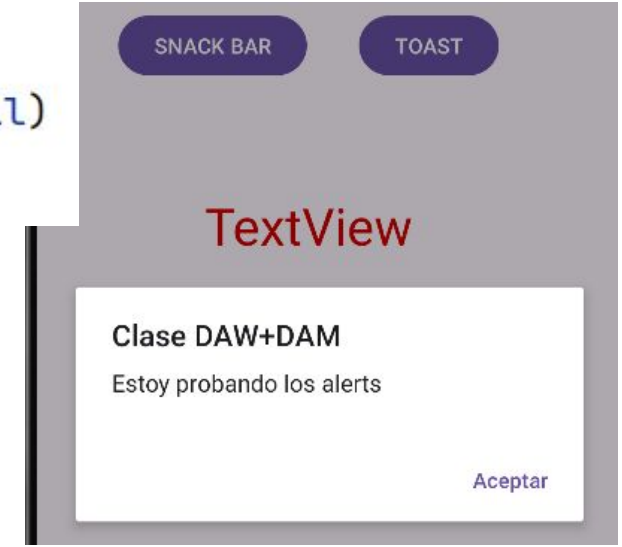
Dentro de un proyecto, en mi caso llamado MyMensajesUsuario, en la actividad añadimos 2 botones, uno para sacar el AlertDialog que viene “de caja” y otro en el que sacaremos un AlertDialog personalizado:

En el click del primer botón incluimos el código que consiste en:

- Instanciar la clase AlertDialog.builder en una variable pasándole el contexto del Activity actual.
- Seguidamente se le da el contenido del título y el mensaje a mostrar y un botón con el texto de “Aceptar”
- Una vez lleno, se crea el propio AlertDialog a partir de la variable que tenías y se muestra con .show().

AlertDialog - Un botón de confirmación

```
var myAlert = AlertDialog.Builder( context: this)
myAlert.setTitle("Clase DAW+DAM")
myAlert.setMessage("Estoy probando los alerts")
myAlert.setPositiveButton( text: "Aceptar",  listener: null)
myAlert.create().show()
```



AlertDialog - Más botones

Para añadir más botones en el AlertDialog (pueden añadirse hasta 3), simplemente debemos indicar el texto (ahora hablaremos de las acciones)

```
myAlert.setNegativeButton( text: "Rechazar", listener: null)
myAlert.setNeutralButton( text: "No me entero", listener: null)
```

AlertDialog - Más botones

Cuando el texto de los botones de demasiado largo para entrar en pantalla, él mismo se dispone de manera vertical.

AlertDialog - Añadimos lógica en los botones

Una vez tenemos los botones, lo normal es que estos hagan algo, especialmente si hay más de 1.

Cada vez que hemos añadido un botón con el método set[Positive/Negative/Neutral]Button, este tenía 2 variables, la primera el texto del botón y la segunda que hemos puesto a “null”.

En esta segunda se codifica la lógica que queramos implementar en los botones. Para ellos debemos llamar al escuchador **DialogInterface.OnClickListener()** y dentro codificamos el método “onClick”.

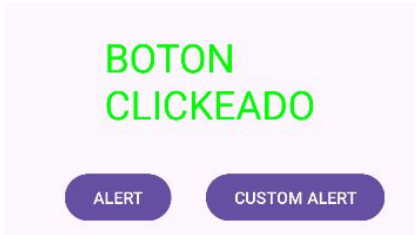
Yo, al igual que he hecho en otras ocasiones, lo he desarrollado implementando la interfaz DialogInterface.OnClickListener(), pero se pueden codificar los métodos directamente si lo preferís.

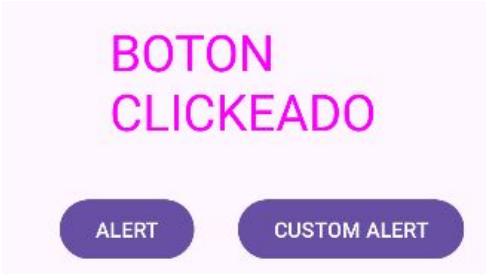
AlertDialog - Añadimos lógica en los botones

```
var myAlert = AlertDialog.Builder( context: this)
myAlert.setTitle("Clase DAW+DAM")
myAlert.setMessage("Estoy probando los alerts")
myAlert.setNeutralButton( text: "No me entero",DialogInterface.OnClickListener( function: this))
myAlert.setNegativeButton( text: "Rechazar",DialogInterface.OnClickListener( function: this))
myAlert.setPositiveButton( text: "Aceptar", DialogInterface.OnClickListener( function: this))
myAlert.create().show()
```

AlertDialog - Añadimos lógica en los botones

```
override fun invoke(p1: DialogInterface, p2: Int) {
    myTextView.text = "BOTON CLICKEADO"
    myTextView.setTextSize(34.0F)
    when (p2) {
        -1 -> {myTextView.setTextColor(Color.BLUE)}
        -2 -> {myTextView.setTextColor(Color.GREEN)}
        -3 -> {myTextView.setTextColor(Color.MAGENTA)}
    }
}
```





AlertDialog - Otras opciones (JAVA)

Además de los botones, podemos añadir una lista de opciones. Si la lista es “fija” y conocida, el primer paso es añadir (como hicimos para el spinner) en el fichero string.xml añadir un array con las opciones a añadir. En mi caso he añadido una lista de deportes:

AlertDialog - Añadiendo una lista de opciones - Paso 2

Luego, en lugar de añadir botones hay que añadir la lista con el método setItems (comento las instrucciones para añadir botones, OJO comento también el mensaje y añado esta lista)

```
MyAlert.setItems(R.array.deportes, new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        // i indica la posición del array clickeada
        pinta_deporte(i);
    }});
```

AlertDialog - Añadiendo una lista de opciones - Paso 3

En mi caso el click sobre cualquier opción me lleva a un método que saca un toast con el deporte elegido

```
private void pinta_deporte(int i) {
    Context context = getApplicationContext();
    String  text = null;
    switch (i){
        case 0:{
            text = "Usted eligió la opcion 1. Futbol";
            break;
        }
        case 1:{
            text = "Usted eligió la opcion 2. Futbolin";
            break;
        }
        case 2:{
            text = "Usted eligió la opcion 3. Futbol Sala";
            break;
        }
        case 3:{
            text = "Usted eligió la opcion 4. Futbol 7";
            break;
        }
    }
    int duration = Toast.LENGTH_LONG;
    Toast toast = Toast.makeText(context, text, duration);
    toast.show();
}
```

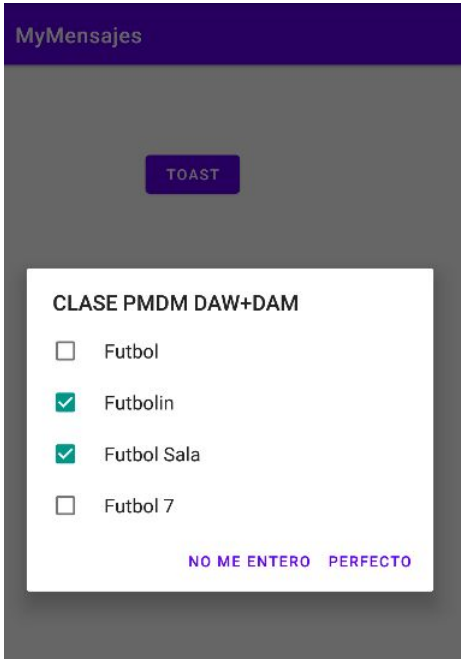
AlertDialog - Añadiendo una lista de opciones

Si la lista de opciones a añadir no es fija y conocida, sino que está en una base de datos, por ejemplo, en lugar de setItems hay que utilizar un adaptador y métodos distintos de carga. No lo vamos a ver.

AlertDialog - Añadiendo una lista de opciones multiselección

También podemos añadir una lista de opciones en las que el usuario tiene que hacer una o varias elecciones y luego aceptar. Para esto se usan los métodos usa los métodos setMultiChoiceItems() o setSingleChoiceItems(), respectivamente, en lugar del setItems.

A la vez puedes tener los botones positivos, negativos, etc, y en el click del botón codificar el escuchador para hacer lo que sea en función de las opciones seleccionadas.



AlertDialog - Personalizamos el AlertDialog

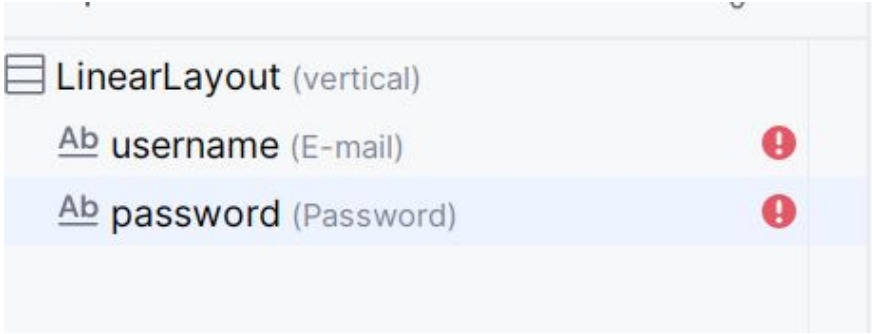
También se puede personalizar el contenido del AlertDialog. Para esto, y es algo que haremos más veces dentro del curso. definiremos un layout XML con los elementos a mostrar en el diálogo.

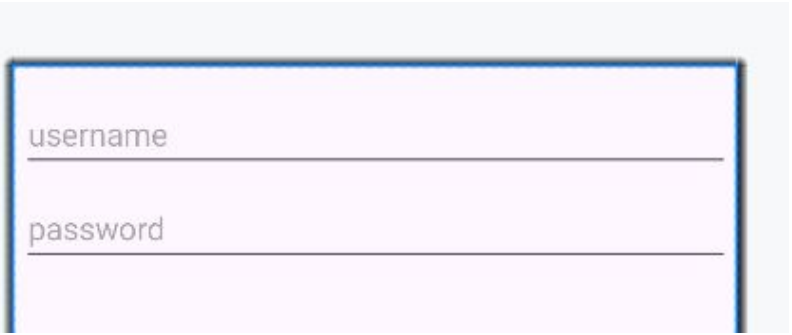
Dentro de la carpeta res / layout creamos un nuevo layout, que en mi caso he llamado login2.xml.

Dentro de este login2.xml he puesto un layout vertical y 2 editText para incluir el usuario y la password.

Subo el layout a la carpeta de recursos de la unidad, podeis bajarlo y copiarlo

AlertDialog - Personalizamos el AlertDialog





AlertDialog - Personalizamos el AlertDialog

De forma predeterminada, el diseño personalizado ocupa toda la ventana de diálogo, pero aún puedes usar métodos AlertDialog.Builder para agregar botones y un título (el texto del Alter no tiene sentido)

En primer lugar y a nivel de clase (¿porqué a nivel de clase? luego lo veremos) definimos un objeto de View que será sobre el que inflemos el layout.

```
lateinit var mybt_alert: Button
lateinit var mybt_customalert: Button
lateinit var myViewInflado: View
```

AlertDialog - Personalizamos el AlertDialog

Siempre que queremos añadir un layout a alguna actividad en tiempo de ejecución habrá que “inflar” el layout. Esto se hace con la clase **layoutInflater** y su método **inflate.**

Luego debemos asignar *la vista inflada* al Alert para que se vea

```
var myAlert = AlertDialog.Builder( context: this)
val myinflater = layoutInflater
myViewInflado=  myinflater.inflate(R.layout.login2,  root: null)
myAlert.setView(myViewInflado)
```

AlertDialog - Personalizamos el AlertDialog

El resto de construcción del Alert, añadiendo botones (en este caso sólo añadiré 2 botones)y mostrándolo, no cambia:

```
myAlert.setNegativeButton( text: "Rechazar",DialogInterface.OnClickListener( function: this))
myAlert.setPositiveButton( text: "Aceptar", DialogInterface.OnClickListener( function: this))
myAlert.create().show()
```

Tampoco cambia mucho el método salvo, y esto es importante, el acceso a los campos de Login y Password

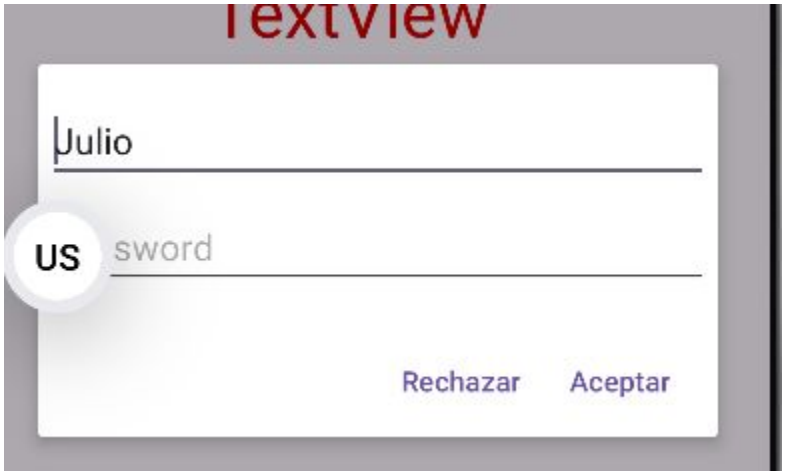
AlertDialog - Personalizamos el AlertDialog

Para acceder a los EditText de usuario y password debemos tener en cuenta que NO estamos en la actividad principal, estamos en otra actividad (el propio alert) que se ha construido con nuestra vista inflada.

Por tanto el acceso a estos EditText se hace a partir de la propia vista informada. Yo voy a coger sólo el usuario, pero la password se haría exactamente igual

```
var myNombre = myViewInflado.findViewById<EditText>(R.id.username)
myTextView.text = myNombre.text.toString()
myTextView.setTextSize(34.0F)
when (p2) {
    -1 -> {myTextView.setTextColor(Color.BLUE)}
    -2 -> {myTextView.setTextColor(Color.GREEN)}
}
```

AlertDialog - Personalizamos el AlertDialog



Notificaciones en Android: Otras notificaciones

Además de los mensajes Toast y los AlertDialog, hay otro tipo de notificaciones en Android Studio como son la Barra de Estado.

No están contenidas dentro de la programación del curso, pero dejo documentación donde se explican cómo implementar su uso:

https://www.sgoliver.net/blog/notificaciones-en-android-ii-barra-de-estado/

Documentación

https://developer.android.com/guide/topics/ui/dialogs?hl=es-419

https://mscdroidlabs.es/como-crear-un-dialogo-alertdialog/

https://www.sgoliver.net/blog/notificaciones-en-android-iii-dialogos/