2025

Programación Multimedia y Dispositivos Móviles (PMDM)

[ANDROID: COMPONENTE DIALOG]

Tabla de contenido

1. Introducción

Un diálogo es una ventana pequeña que le indica al usuario que debe tomar una decisión o ingresar información adicional.

No ocupa toda la pantalla y, por lo general, se usa para eventos modales que requieren que los usuarios realicen alguna acción para poder continuar.

La clase base para los diálogos es **Dialog**, pero no se crean instancias de Dialog, sino de alguna de sus subclases:

- **AlertDialog**: es un diálogo que puede mostrar un título, hasta tres botones, una lista de elementos seleccionables o un diseño personalizado.
- **DatePickerDialog ó TimePickerDialog**: es un diálogo con una interfaz predefinida que le permite al usuario seleccionar una fecha o una hora.

2. AlertDialog

Esta clase permite crear una variedad de diseños de diálogo y, a menudo, es la única clase de diálogo que se necesita.

Hay tres regiones en un diálogo de alerta:

- **Título**: es opcional y sólo se usa cuando el área de contenido está ocupada por un mensaje detallado, una lista o un diseño personalizado. Para indicar un mensaje o una pregunta simple, no se necesita un título. El título se establece con el método **setTitle()**.
- **Área de contenido**: puede mostrar un mensaje, una lista o cualquier otro diseño personalizado.
- **Botones de acción**: puede haber hasta tres botones de acción en un diálogo.

La clase **AlertDialog.Builder** proporciona APIs que permiten crear un AlertDialog con estos tipos de contenido.

El método **create()** crea un objeto de tipo AlertDialog. Y el método **show()** muestra el diálogo.

Ejemplo:

```
val builder: AlertDialog.Builder = AlertDialog.Builder(context)
builder
      .setMessage("Esto es el mensaje")
      .setTitle("Esto es el título")
val dialog: AlertDialog = builder.create()
dialog.show()
```

3. Cómo agregar botones de acción

Hay tres botones de acción que se pueden agregar:

- **Positivo**: usado para aceptar y continuar con la acción “Aceptar”.
- **Negativo**: usado para cancelar la acción.
- **Neutral**: usado cuando no se quiere continuar con la acción, pero tampoco se quiere cancelar. Aparece entre los botones positivo y negativo. Por ejemplo: la acción puede ser “Recordar más tarde”.

Sólo se puede agregar un botón de cada tipo. Es decir, no puede haber más de un botón positivo.

Ejemplo:

```
val builder: AlertDialog.Builder = AlertDialog.Builder(context)
builder
      .setMessage(“Esto es el mensaje”)
      .setTitle(“Esto es el título)
      .setPositiveButton(“Aceptar”){ dialog, which ->
            // Tareas a realizar
      }
      .setNegativeButton(“Cancelar){ dialog, which ->
            // Tareas a realizar
      }
val dialog: AlertDialog = builder.create()
dialog.show()
```

4. Cómo agregar una lista

Hay tres tipos de listas disponibles con las APIs de AlertDialog:

- Una lista de opción única tradicional
- Una lista de opción única persistente (botones de selección)
- Una lista de opciones múltiples persistente (casillas de verificación)

Crear una lista de opción única tradicional

La lista aparece en el área de contenido del diálogo, por lo tanto, no se puede mostrar un mensaje y una lista al mismo tiempo.

Para especificar los elementos de la lista se utiliza el método **setItems()** pasando un **array**. Como alternativa, se puede especificar una lista con el método **setAdapter()**, lo que permite cargar la lista con datos dinámicamente usando un **ListAdapter**. Si se carga con el ListAdapter siempre hay que usar un **Loader** de modo que el contenido se cargue de forma asíncrona.

De forma predeterminada, cuando se presiona un elemento de la lista se cierra el diálogo, a menos que se use una de las opciones de lista persistente.

Ejemplo:

```
val builder: AlertDialog.Builder = AlertDialog.Builder(context)
builder
      .setTitle("Esto es el título)
      .setPositiveButton("Aceptar"){ dialog, which ->
            // Tareas a realizar
      }
      .setNegativeButton("Cancelar){ dialog, which ->
            // Tareas a realizar
      }
      .setItems(arrayOf("Item 1", "Item 2", "Item 3")){ dialog, which ->
            // Tareas a realizar
      }
val dialog: AlertDialog = builder.create()
dialog.show()
```

Crear una lista persistente de opción única o de opciones múltiples

Para crear una lista de elementos de varias opciones (casillas de verificación) o de una opción (botones de opción) hay que usar los métodos **setMultiChoiceItems()** o **setSingleChoiceItems()** respectivamente.

Ejemplo de lista de opciones múltiples:

```
val builder: AlertDialog.Builder = AlertDialog.Builder(context)
builder
      .setTitle("Esto es el título)
      .setPositiveButton("Aceptar"){ dialog, which ->
            // Tareas a realizar
      }
      .setNegativeButton("Cancelar){ dialog, which ->
            // Tareas a realizar
      }
      .setMultiChoiceItems(
            arrayOf("Item 1", "Item 2", "Item 3"), null){ dialog, which,
            isChecked ->
                  // Tareas a realizar
      }
val dialog: AlertDialog = builder.create()
dialog.show()
```

El método **setMultiChoiceItems()** recibe como **parámetros** un ***array o lista*** con los elementos y un ***array o lista de booleanos*** que indica los ítems seleccionados por defecto al mostrar el diálogo.

Se ejecuta la función lambda del método **setMultiChoiceItems()** cada vez que se seleccione uno de los ítems siendo ***which*** el **índice del elemento en el array** (para el "Item 1" which será 0, para el "Item 2" which será 1, …) y ***isChecked*** un booleano que indicará si ese ítem ha sido seleccionado o no.

Ejemplo de lista de opción única:

```
val builder: AlertDialog.Builder = AlertDialog.Builder(context)
builder
      .setTitle("Esto es el título)
      .setPositiveButton("Aceptar"){ dialog, which ->
            // Tareas a realizar
      }
      .setNegativeButton("Cancelar){ dialog, which ->
            // Tareas a realizar
      }
      .setSingleChoiceItems(
            arrayOf("Item 1", "Item 2", "Item 3"), 0){ dialog, which ->
                  // Tareas a realizar
      }
val dialog: AlertDialog = builder.create()
dialog.show()
```

El método **setSingleChoiceItems()** recibe como **parámetros** un ***array o lista*** con los elementos y un ***número*** que será la posición que ocupa en el array el ítem que se desea aparezca seleccionado al mostrar el diálogo.

5. Cómo crear un diseño personalizado

Para crear un diseño personalizado en un diálogo debes crear el diseño y agregárselo a un AlertDialog llamando al método **setView()** del **objeto AlertDialog.Builder**.

De forma predeterminada el diseño personalizado ocupa toda la ventana de diálogo, pero se pueden usar los métodos del objeto AlertDialog.Builder para agregar botones y un título.

Ejemplo de diseño:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <ImageView
        android:src="@drawable/header_logo"
        android:layout_width="match_parent"
        android:layout_height="64dp"
        android:scaleType="center"
        android:background="#FFFFBB33"
        android:contentDescription="@string/app_name" />
    <EditText
        android:id="@+id/username"
        android:inputType="textEmailAddress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="4dp"
        android:hint="@string/username" />
```

```
            <EditText
                android:id="@+id/password"
                android:inputType="textPassword"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="4dp"
                android:layout_marginLeft="4dp"
                android:layout_marginRight="4dp"
                android:layout_marginBottom="16dp"
                android:fontFamily="sans-serif"
                android:hint="@string/password"/>
        </LinearLayout>
```

Como contenedor del diálogo se necesita un **DialogFragment**. Esta clase proporciona todos los controles necesarios para crear el diálogo y administrar su apariencia. Para utilizar el diseño personalizado hay que utilizar el método **inflate()** del **LayoutInflater**. El primer parámetro es el ID del recurso de diseño (layout) y el segundo es una vista superior para el diseño. Por último, se llama al método **setView()** para colocar el diseño en el diálogo.

Ejemplo:

```
class MyDialogFragment: DialogFragment(){
  override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
      return activity?.let {
            val builder = AlertDialog.Builder(it)
            // obtiene el layout inflater.
            val inflater = requireActivity().layoutInflater

            // Coloca el diseño en el diálogo.
            // La vista padre es null porque va en el diseño del diálogo
            builder.setView(inflater.inflate(R.layout.dialog_signin, null))
                  // Se añaden los botones de acción
                  .setPositiveButton(R.string.signin,
                        DialogInterface.OnClickListener { dialog, id ->
                              // Realizar tareas.
                        })
                  .setNegativeButton(R.string.cancel,
                        DialogInterface.OnClickListener { dialog, id ->
                              getDialog().cancel()
                        })
            builder.create()
      } ?: throw IllegalStateException("La actividad no puede ser null")
  }
}
```

6. Cómo mostrar un diálogo

Para mostrar tu diálogo personalizado hay que crear una **instancia de** la clase **DialogFragment** y llamar al **método show()** pasándole como **parámetros** el **FragmentManager** y un **tag** o etiqueta.

El ***FragmentManager*** se obtiene con la propiedad **supportFragmentManager** accesible desde la actividad.

El ***tag o etiqueta*** es un **nombre único** que el sistema usa para guardar y restaurar el estado del fragmento cuando sea necesario. También permite obtener un controlador para el fragmento llamando al método **findFragmentByTag()**.

Ejemplo:

```
DialogoPersonalizadoFragment().show(supportFragmentManager, "Diálogo
Personalizado")
```

7. Cómo mostrar una actividad como un diálogo

Se puede mostrar una Activity como un diálogo en lugar de usar las APIs de Dialog. Para ello, crea una actividad y establece su tema en **Theme.Holo.Dialog** en el elemento <activity> del AndroidManifest.xml.

Ejemplo:

AndroidManifest.xml

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

ActividadDialogo.kt

```
class ActividadDialogo: Activity(){
        override fun onCreate(savedInstanceState: Bundle?){
                super.onCreate(savedInstanceState)
                this.setContentView(R.layout.disenio_dialogo)
        }
}
```

Ahora la actividad se muestra en una ventana de diálogo en lugar de mostrarse en pantalla completa.

8. Cómo descartar un diálogo

Cuando el usuario presiona un botón de acción creado con un AlertDialog.Builder, el sistema descarta el diálogo.
Cuando el usuario presiona un elemento de una lista de diálogo, excepto si la lista utiliza botones de selección o casillas de verificación, el sistema descarta el diálogo.

Para descartar el diálogo manualmente hay que llamar al método **dismiss()** del DialogFragment.

Si es necesario realizar acciones cuando el diálogo desaparezca, entonces hay que implementar el método **onDismiss()** del DialogFragment.

También se puede cancelar un diálogo. Este evento es especial e indica que el usuario abandonará el diálogo sin completar la tarea. Esto ocurre cuando se presiona el botón Atrás o la pantalla de fuera del área del diálogo y cuando se llama explícitamente al método **cancel()** en el Dialog (en respuesta al botón “Cancelar” en el diálogo).
Se puede responder al evento de cancelación implementando el método **onCancel()** en la clase DialogFragment.

Cuando se llama a cancel(), el sistema llama a dismiss(); pero si se llama a dismiss() no se ejecuta cancel().

Por lo general, se llama a **dismiss()** cuando el usuario presiona el **botón positivo** en el **diálogo** para **quitar**lo **de la vista**.