Diálogos

Daniel Gómez Benítez

Curso: 2º Desarrollo de aplicaciones multiplataforma

Asignatura: Programación multimedia y dispositivos móviles

Profesor: Eva María Palomo Carcamo

Índice

Autores..... 3
Introducción..... 4
Cuadros de diálogo.....5
AlertDialog.....5
DatePickerDialog/TimePickerDialog..... 5
Cómo crear Fragmentos de Diálogo..... 6
Crear Diálogo de Alerta.....7
Crear Diálogo con progreso..... 7
Crear Diálogo con múltiples opciones.....8
Crear diálogo añadiendo una lista tradicional..... 9
Crear diálogo personalizado con obtención de datos..... 10
Crear diálogo para selección de fecha.....13
Crear diálogo para selección de tiempo.....14
Bibliografía..... 15

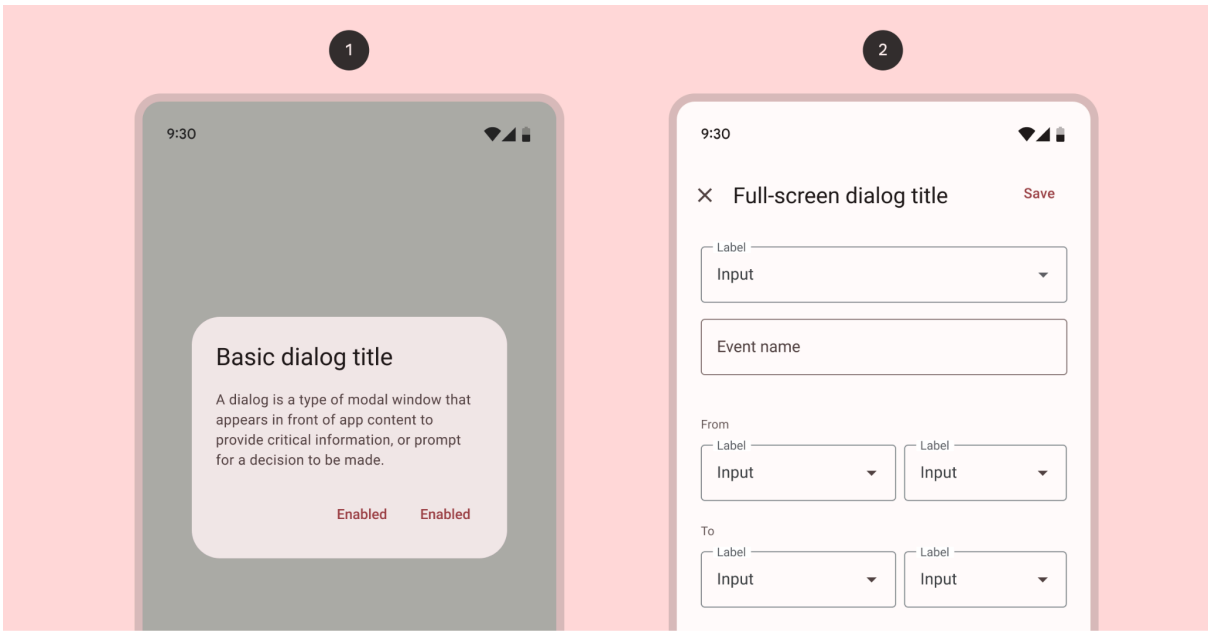
Autores

Daniel Gómez Benítez

Cuadros de diálogo

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

Existen 2 tipos: Básico y pantalla completa



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/TimePickerDialog

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

Para crear diálogos tendremos que llamar a la clase que extiende de Dialogs, si queremos usar una ventana de diálogo normal, tendremos que poner AlertDialog.Builder ventanaDialog = new AlertDialog.Builder(context);

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

Cómo crear Fragmentos de Diálogo

Puedes lograr una gran variedad de diseños de diálogo, incluidos diseños personalizados y los que se describen en la guía de diseño Diálogos, si extiendes DialogFragment y creas un AlertDialog en el método de devolución de llamada onCreateDialog().

Por ejemplo, a continuación, hay un AlertDialog básico administrado dentro de un DialogFragment:

```
public class DialogFragmentPersonalizado extends DialogFragment {
   @Override
   public Dialog onCreateDialog(Bundle savedInstanceState) {
      AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
      builder.setMessage(“Comenzar juego”)
            .setPositiveButton(“Comenzar”, new DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int id) {
               }
            })
            .setNegativeButton(“Cancelar”, new DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int id) {
               }
            });
      return builder.create();
   }
}
```

Crear Diálogo de Alerta

La clase AlertDialog te permite crear una variedad de diseños de diálogo y generalmente es la única clase de diálogo que necesitarás.
Existen 3 regiones en el cuadro de diálogo de alerta:

1. Título
Es opcional y solo se debe usar cuando el área de contenido está 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.

Crear Diálogo con progreso

Para mostrar una carga sobre un tipo de acción.
Dentro de R.layout.dialog_pb está la personalización con el progress bar.

```
pbDialog.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
       AlertDialog.Builder builder = new
AlertDialog.Builder(MainActivity.this);
       builder.setView(R.layout.dialog_pb);

   AlertDialog alertDialog = builder.create();
   alertDialog.setMessage("Cargando datos");
   alertDialog.show();
   }
});
```

Crear Diálogo con múltiples opciones

Sirve para crear una ventana con varias opciones seleccionables para guardar información del usuario, por ejemplo para saber que frutas le gustan más.

Clase fragmento de diálogo

```
public class DialogMultiplesOpciones extends DialogFragment {
  @Override
   public Dialog onCreateDialog(Bundle savedInstanceState){
      String[] arrayFrutas = {"Manzana", "Plátano"};
      AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

      builder.setMessage("Frutas");

      builder.setMultiChoiceItems(arrayFrutas, null, new
DialogInterface.OnMultiChoiceClickListener(){
          @Override
          public void onClick(DialogInterface dialogInterface, int i,
boolean isChecked){
               //Lógica para interactuar con los objetos seleccionados
          }
      });

      builder.setPositiveButton("Aceptar", new
DialogInterface.OnClickListener() {
          @Override
          public void onClick(DialogInterface dialogInterface, int i) {
               //Para guardar múltiples datos, en este caso las frutas que
he seleccionado
          }
      });

      builder.setPositiveButton("Cancelar", new
DialogInterface.OnClickListener() {
          @Override
          public void onClick(DialogInterface dialogInterface, int i) {
              //Cancela y cierra la ventana
          }
      });

      return builder.create();
  }
```

Clase main

```
dialogMO.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
       DialogMultiplesOpciones dialogMO = new DialogMultiplesOpciones();
       dialogMO.show(getSupportFragmentManager(), "DialogRadio");
   }
});

dialogLista.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
       DialogLista dialogLista = new DialogLista();
       dialogLista.show(getSupportFragmentManager(), "DialogLista");
   }
});
```

Crear diálogo añadiendo una lista tradicional

Lo mismo que el ejemplo anterior pero solo para visualizar una simple lista de diferentes objetos, su realización es mucho más simple.

Clase fragmento de diálogo

```
public class DialogLista extends DialogFragment {

   @Override
  public Dialog onCreateDialog(Bundle savedInstanceState){
       AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
       builder.setTitle("Lista compra");
       builder.setItems(R.array.Frutas, new
DialogInterface.OnClickListener() {
           @Override
           public void onClick(DialogInterface dialogInterface, int i) {

           }
       });
       return builder.create();
   }
}
```

Clase Main

```
dialogLista.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
       DialogLista dialogLista = new DialogLista();
       dialogLista.show(getSupportFragmentManager(), "DialogLista");
   }
});
```

Crear diálogo personalizado con obtención de datos

En este ejemplo se personalizará la interfaz de un diálogo a tu gusto y a tu manera, escribiendo datos desde el fragmento y obteniéndose desde el MainActivity, lo primero es crear el diálogo y llamarlo desde el Main.

Clase fragmento de diálogo

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState){
   AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
   LayoutInflater inflator = requireActivity().getLayoutInflater();
   View view = inflator.inflate(R.layout.custom_dialog, null);
   builder.setView(view);
   username = view.findViewById(R.id.username);
   password = view.findViewById(R.id.password);
   builder.setPositiveButton("Iniciar sesión", new
DialogInterface.OnClickListener() {
       @Override
       public void onClick(DialogInterface dialog, int id) {
           if(username.getText().toString().length() > 0){
               String user = username.getText().toString();
               String pass = password.getText().toString();

           }
       }
   });
           builder.setNegativeButton("Cancelar", new
DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int id) {
               }
           });

   return builder.create();
}
```

Clase Main

```
dialogCustom.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        DialogPersonalizado dialogPersonalizado = new DialogPersonalizado();
        dialogPersonalizado.show(getSupportFragmentManager(), "Dialog
personalizado");
    }
});
```

Ahora que tenemos el diálogo hecho y pudiendo escribir datos, lo que queremos es obtenerlos en la clase Main, usaremos una interfaz para la comunicación.
En la clase DialogPersonalizado, creamos una interfaz llamada DialogListener.

```
public interface DialogListener{
    void aplicarTextos(String username, String password);
}
```

En ella se encuentra el método que se necesita para la obtención de los datos.

Para que exista la conexión usaremos el método de la clase Fragment, onAttach, el cual recibe un contexto, ese contexto es la actividad a la que hemos referenciado anteriormente, en la creación del AlertBuilder, donde pusimos getActivity(), ahora solo falta conectar la interfaz con el Main para que el fragmento y el Main estén conectados

```
@Override
public void onAttach(Context context){
    super.onAttach(context);
    try {
        listener = (DialogListener) context;
        // Intenta realizar un casting del contexto (context) a un objeto
que implemente la interfaz DialogListener.
        // Esto significa que esperas que la actividad que contiene el
fragmento implemente la interfaz DialogListener.
    } catch (ClassCastException e) {
        throw new ClassCastException(context.toString() + " debe implementar
DialogListener");
    }
}
```

El try catch se usa por si la interfaz no se llega a implementar en la clase Main, daría error.

Después, implementamos la interfaz en la clase Main y declaramos el método aplicarTextos para así obtener la información que se le pase a esos parámetros cuando el método sea llamado, en este caso como la información proviene del fragmento, el método se llamará desde ahí, pasando como parámetro el nombre de usuario y contraseña que hemos proporcionado, así desde la clase Main se podrá obtener la información.

Declaración del método aplicarTextos

```
public void aplicarTextos(String nombre, String contrasena){
   username.setText(nombre);
   password.setText(contrasena);
}
```

Método onCreateDialog actualizado

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState){
   AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
   LayoutInflater inflator = requireActivity().getLayoutInflater();
   View view = inflator.inflate(R.layout.custom_dialog, null);
   builder.setView(view);
   username = view.findViewById(R.id.username);
   password = view.findViewById(R.id.password);
   builder.setPositiveButton("Iniciar sesión", new
DialogInterface.OnClickListener() {
       @Override
       public void onClick(DialogInterface dialog, int id) {
           if(username.getText().toString().length() > 0){
               String user = username.getText().toString();
               String pass = password.getText().toString();
               listener.aplicarTextos(user, pass);
           }
       }
   });
```


Alumno: Daniel Gómez Benítez

Crear diálogo para selección de fecha

Este diálogo sirve para la obtención de información de la fecha que seleccione el usuario. Esto puede servir para diferentes funciones por ejemplo la fecha en la que el usuario quiere recibir un recordatorio.

```
dialogDate.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Calendar calendar = Calendar.getInstance();
        int anio = calendar.get(Calendar.YEAR);
        int mes = calendar.get(Calendar.MONTH);
        int day = calendar.get(Calendar.DAY_OF_MONTH);

        DatePickerDialog datePickerDialog = new DatePickerDialog(
                MainActivity.this,
                new DatePickerDialog.OnDateSetListener() {
                    @Override
                    public void onDateSet(DatePicker view, int year, int
month, int dayOfMonth) {
                        // Aquí puedes manejar la fecha seleccionada
                         String fechaSeleccionada = dayOfMonth + "/" + (month
+ 1) + "/" + year;
                        Toast.makeText(MainActivity.this, "Fecha
seleccionada: " + fechaSeleccionada, Toast.LENGTH_SHORT).show();
                    }
                }, anio, mes, day
        );
        datePickerDialog.show();
    }
});
```

Con anio, mes y day, obtendremos el día actual con el que se inicia seleccionado el calendario.

Crear diálogo para selección de tiempo

Este diálogo sirve para que el usuario pueda seleccionar una hora y minuto, esto puede servir para diferentes funciones por ejemplo la selección del tiempo para una alarma.

```
dialogTime.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View view) {
           Calendar calendar = Calendar.getInstance();
           int hora = calendar.get(Calendar.HOUR);
           int minutos = calendar.get(Calendar.MINUTE);

           TimePickerDialog timePickerDialog = new
TimePickerDialog(MainActivity.this, new TimePickerDialog.OnTimeSetListener()
{
               @Override
               public void onTimeSet(TimePicker timePicker, int i, int i1) {
                   String tiempoSeleccionado = "Has seleccionado: " + i + ":"
+ i1;
                   Toast.makeText(MainActivity.this, tiempoSeleccionado,
Toast.LENGTH_SHORT).show();
               }
           },hora,minutos, true // false para usar el formato de 24 horas,
true para usar el formato de 12 horas
           );
           timePickerDialog.show();
       }
   });
}
```

hora y minutos es el tiempo actual.

Bibliografía

[Información diálogos]