Desarrollo para dispositivos móviles con Android: diálogos.

Contenido

1 Introducción

En ocasiones, es necesario interactuar con el usuario para perfilar una acción en detalle. Por ejemplo, es válido preguntarle al usuario si está seguro de querer borrar un archivo. Es incluso aconsejable informar al usuario de que se está procediendo a realizar una tarea que puede llevar un cierto tiempo. Para este tipo de necesidades, están disponibles los diálogos.

Es importante tener en cuenta que los diálogos en Android no son síncronos, sino asíncronos. Es decir, una vez se llama al método *show()*, la ejecución no se detiene hasta completar el diálogo, sino que continúa normalmente. Por ello, es en los gestores de eventos donde deben guardarse la información obtenida del usuario.

2 Toasts - Creando notificaciones

En ocasiones, es necesario avisar al usuario de que se va a realizar una tarea, o de que ya se ha realizado, sin que sea necesario esperar a que pulse un botón o similar para continuar. Simplemente, el mensaje aparece unos segundos en pantalla y desaparece. En terminología Android, este tipo de mensaje es un *toast*, nunca tiene el foco, la actividad sigue pudiendo ser manejada, y solamente ocupa aquel espacio en pantalla que sea imprescindible.

Crear un toast básico es muy sencillo. Se trata de indicar la actividad sobre la que va a aparecer, el texto de la notificación, y la duración. El método estático **Toast**.*makeText()*, que permite crear el *toast* especificando dichos parámetros. Una vez creado, puede ser mostrado mediante su método *show()*. En cuanto a la duración, se puede escoger entre corta (**Toast.LENGTH_SHORT**) y larga (**Toast.LENGTH_LONG**).

```
Toast.makeText( this, “Notificado estás”, Toast.LENGTH_SHORT ).show();
```

Por supuesto, estas dos acciones pueden dividirse en dos partes si se desea.

```
Toast t = Toast.makeText( this, “Notificado quedas”, Toast.LENGTH_SHORT );
t.show();
```

Finalmente, las *toasts* se pueden posicionar usando el método *setGravity()*. Este método acepta un valor que se puede componer con las constantes de la clase **Gravity**. Por ejemplo, más abajo se crea un *toast* en la parte superior izquierda de la pantalla. Los valores subsiguientes son *offsets* en horizontal y vertical, respectivamente.

```
Toast t = Toast.makeText( this, “Notificado quedas”, Toast.LENGTH_SHORT );
t.setGravity( Gravity.TOP | Gravity.LEFT, 0, 0 );
t.show();
```

3 Interactuando con el usuario con AlertDialog

Las *toasts* son ciertamente limitadas. En ocasiones, es conveniente comprobar que el usuario ha visto el mensaje. En otras ocasiones, es necesario obtener una información complementaria por su parte.

Es de notar que para crear diálogos de alerta es necesario pasar la actividad (**Activity**) de la depende ese diálogo en el constructor (**AlertDialog.Builder**) del mismo, que en este caso aparece como *this*, probablemente el caso más común, al lanzarse el diálogo desde la propia actividad (**MainActivity**). Si se hace desde un gestor de eventos, entonces será necesario sustituirlo por *MainActivity.this*, para de esta forma estar seguro que el diálogo se ejecutará en el contexto

correcto. Por si acaso, en los siguientes ejemplos se cualifica **this** completamente, asumiendo que el nombre dado a la clase correspondiente a la actividad principal es el valor por defecto: **MainActivity**.

3.1 Mensajes

Lo más parecido a una *toast*, que si bien requiere interacción por el usuario para continuar, es un diálogo de alerta como el que sigue, sin botones:

```
AlertDialog.Builder builder = new AlertDialog.Builder( MainActivity.this );
builder.setTitle( "Mi Android App" );
builder.setMessage( "Esta es mi primera App en Android" );
builder.create().show();
```

Lo siguiente es añadirle botones, y reaccionar ante la pulsación de los mismos.

```
AlertDialog.Builder builder = new AlertDialog.Builder( MainActivity.this );
builder.setTitle( "Mi Android App" );
builder.setMessage( "Esta es mi primera App en Android" );
builder.setPositiveButton( "Probarla!", null );
builder.setNeutralButton( "No sé, déjame pensarlo...", null );
builder.setNegativeButton( "Salir!", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dlg, int i) {
                System.exit( 0 );
            }
});
builder.create().show();
```

Como siempre, solamente es necesario reaccionar a los eventos utilizando los *listeners*. En este caso, se trata de los residentes dentro de **DailogInterface**. Los diálogos solo pueden soportar un botón de cada tipo: positivo (como “Ok”), neutral (como “Recuérdamelo más tarde”), o negativo (como “Cancelar”).

3.2 Ofreciendo varias posibilidades

Si son necesarias más acciones, entonces es el momento de probar con una lista.

```
AlertDialog.Builder builder = new AlertDialog.Builder( MainActivity.this );
builder.setTitle( "Letra NIF" );
builder.setItems( new String[]{ "Probarla!", "Salir!" }, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dlg, int opc) {
                if ( opc == 1 ) {
                    System.exit( 0 );
                }
            }
});
builder.create().show();
```

El método *onClick()* indica en su argumento entero la opción que ha sido pulsada. Nótese que si se visualiza un mensaje, no se puede establecer una lista de opciones.

De una forma muy similar se puede establecer una lista de elementos para elegir solamente uno (*radio button*), o para elegir varios (*check box)*. La única diferencia real es que es necesario establecer en el manejador del evento las opciones seleccionadas.

```
this.opc = 0;
AlertDialog.Builder builder = new AlertDialog.Builder( MainActivity.this );
builder.setTitle( "Letra NIF" );

builder.setSingleChoiceItems( new String[]{ "Probarla!", "Salir!" }, 0,
                new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dlg, int op) {
                        Main.this.opc = op;
                    }
                });
builder.setPositiveButton("Ok", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                if ( Main.this.opc == 1 ) {
                    System.exit( 0 );
                }
            }
});
builder.create().show();
```

Lo mismo puede hacerse con *check boxes*, aunque en este caso no se adapte demasiado bien al ejemplo.

```
this.opc = 0;
AlertDialog.Builder builder = new AlertDialog.Builder( MainActivity.this );
builder.setTitle( "Letra NIF" );

builder.setMultiChoiceItems(
                new String[]{ "Probarlo!", "Salir!" },
                new boolean[]{ true, false },
                new DialogInterface.OnMultiChoiceClickListener() {
                    @Override
                     public void onClick(DialogInterface dialogInterface, int i,
boolean b) {
                        if ( b ) {
                            Main.this.opc = i;
                }}});

builder.setPositiveButton("Ok", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                if ( Main.this.opc == 1 ) {
                    System.exit( 0 );
                }
            }
        });
builder.create().show();
```

En el caso de emplear *check boxes*, se pasa un vector de *booleanos* con tantos elementos como se desee representar, de forma que aquellos que coincidan en índice y estén a **true**, aparecerán marcados. El método *OnClick(d, i, b)* de la clase **DialogInterface.OnMultiChoiceClickListener** acepta dos informaciones primordiales (como siempre, *d* es una referencia al **AlertBuilder**): *i*, el número de opción que fue pulsada, y *b*, en qué estado ha quedado (verdadero o falso). En la siguiente sección se discute un ejemplo más detallado.

3.3 Un ejemplo más detallado con selección múltiple

En el siguiente ejemplo, se va a construir una aplicación diseñada para permitir seleccionar los ingredientes de una hamburguesa, e informarnos del coste total de la misma (a partir de un coste base, de carne y pan, de 3€). Los ingredientes disponibles serán lechuga, tomate, queso, york y cebolla. La siguiente clase muestra la lógica de negocio del problema.

```
/** Calculates costs related to a burguer. */
public class BurguerConfigurator {
    public static final double BASE_COST = 3.0;

    public BurguerConfigurator()
    {
        this.selected = new boolean[ INGREDIENTS.length ];

        assert INGREDIENTS.length != COSTS.length:
                "all arrays should have the same length";
    }

    /** Calculates the cost for the burguer
     * @return A real with the total cost. */
    public double calculateCost()
    {
        double toret = BASE_COST;

        for(int i = 0; i < this.selected.length; ++i) {
            if ( this.selected[ i ] ) {
                toret += COSTS[ i ];
            }
        }

        return toret;
    }

    /** Returns the live boolean vector for ingredient selection */
    public boolean[] getSelected()
    {
        return this.selected;
    }

    /** Represents all available ingredients */
    public static String[] INGREDIENTS = new String[] {
            "Lechuga", "Tomate",
            "Queso", "York",
            "Cebolla"
    };

    /** Parallel array to ingredients, representing all costs */
    public static double[] COSTS = new double[] {
            0.1, 0.5, 1, 0.75, 0.1
    };

    private boolean[] selected;
}
```

Los vectores de clase *COSTS* (costes), *INGREDIENTS* (ingredientes), y el vector de instancia *selected*, son paralelos. Esto quiere decir que la posición 0 de *INGREDIENTS* nos da la etiqueta para la lechuga, la posición 0 de *COSTS*, el coste de la lechuga, y la posición 0 de *selected* si esta ha sido seleccionada o no.

La clase **MainActivity** crea el controlador para el botón que lanza el diálogo, y cuyo código está en *showIngredientsDialog()*.

```
public class MainActivity...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
      // Create the burguer configuration
      this.cfgBurguer = new BurguerConfigurator();

      // Create callback for the button allowing to select ingredients
      Button btIngredients = (Button) this.findViewById( R.id.btIngredients );

            btIngredients.setOnClickListener(new View.OnClickListener() {
                  @Override
                  public void onClick(View view) {
                      MainActivity.this.showIngredientsDialog();
                  }
        });
      }

      // más cosas...

      private BurguerConfigurator cfgBurguer;
}
```

El método showIngredientsDialog() se muestra a continuación. Simplemente cea un diálogo que referencia el vector de booleanos en el mismo, de forma que al final se sabe los elementos han sido marcados, y cuáles no.

```
public class MainActivity...
    private void showIngredientsDialog()
    {
        final boolean[] selections = this.cfgBurguer.getSelected();
        AlertDialog.Builder dlg = new AlertDialog.Builder( this );

        dlg.setTitle( this.getResources().
                              getString( R.string.lblIngredientSelection) );
        dlg.setMultiChoiceItems(
                BurguerConfigurator.INGREDIENTS,
                selections,
                new DialogInterface.OnMultiChoiceClickListener() {
                    @Override
                    public void onClick(... dlgIntf, int i, boolean b) {
                        selections[ i ] = b;
                    }
                }
        );

        dlg.setPositiveButton( "Ok", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                MainActivity.this.updateTotals();
            }
        });
        dlg.create().show();
    }
```

```
    private void updateTotals()
    {
        final TextView lblTotal = (TextView) this.findViewById( R.id.lblTotal );
        final TextView lblSelection =
                  (TextView) this.findViewById( R.id.lblIngredientsSelected );

        lblTotal.setText( Double.toString(
                              MainActivity.this.cfgBurguer.calculateCost() ) );
        lblSelection.setText( "—" + this.cfgBurguer.toListWith( "\n—" ) );
    }

    // más cosas...
}
```

Los fuentes completos de este ejemplo pueden consultarse o descargarse en *GitHub*[1].

3.4 Toma de datos

No es conveniente abusar de los diálogos: en general, es preferible utilizar una **Activity** en su lugar en cuanto la compleijidad avance lo más mínimo, como se verá más adelante. Sí es posible, en cambio, tomar datos del usuario de esta forma en algunas circunstancias, o simplemente crear una *layout* al gusto.

Así, se está usando en este caso la característica de los diálogos según la cual es posible asignar cualquier *layout* a la porción entre el mensaje y los botones, mediante *setView()*. Si bien se puede especificar un *layout* complejo con contenedores como **LinearLayout**, también es posible especificar una entrada de texto simple (**EditText**) para capturar los datos.

```
AlertDialog.Builder builder = new AlertDialog.Builder( MainActivity.this );
final EditText edUsr = new EditText( this );

this.usr = “”;
builder.setTitle( "Acceder" );
builder.setMessage( "Nombre de usuario" )
builder.setView( edUsr );
builder.setPositiveButton( "Create", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface di, int i) {
                this.usr = edUsr.getText().toString();
            }
});

builder.setNegativeButton( "Cancel", null );
builder.create().show();
```

Una vez leídos datos (en este caso en el atributo *usr*), se pueden tratar en la aplicación. En este ejemplo, el atributo *usr* de la clase **MainActivy** es el que se empleará para guardar el *login* del usuario.

1 https://github.com/baltasarq/BurguerBuilder/

4 Diálogos para escoger fecha y hora

Los diálogos para escoger fecha (**DatePickerDialog**) y hora (**TimePickerDialog**) solo permiten realizar dichas acciones, y por tanto no necesitan un constructor para crearlos, utilizando directamente el constructor de la propia clase. Nótese que en dicho constructor, *this* se refiere a una actividad, en este caso la misma que lanza el diálogo (**Main**). Las fechas siempre se proporcionan como año, mes, día.

```
DatePickerDialog dlg = new DatePickerDialog(
                MainActivity.this,
                new DatePickerDialog.OnDateSetListener() {
                    @Override
                    public void onDateSet(DatePicker dp, int y, int m, int d) {
                              Main.this.year = y;
                              Main.this.month = m;
                              Main.this.day = d;
                    }
                },
                2015, 10, 4
);
dlg.show();
```

El diálogo para escoger una hora es similar, solo que no se indican los segundos, y un booleano indica si se desea formato de fecha de 12 o 24 horas (*true*).

```
TimePickerDialog dlg = new TimePickerDialog(
                MainActivity.this,
                new TimePickerDialog.OnTimeSetListener() {
                    @Override
                    public void onTimeSet(TimePicker dp, int h, int m) {
                              Main.this.hour = h;
                              Main.this.minute = m;
                    }
                },
                11, 01, true
);
dlg.show();
```

5 Referencias

- Documentación y recursos de Android para desarrolladores (accedido en sept. 2015) http://developer.android.com/
- Tutoriales oficiales de Android https://developer.android.com/training/
- Toasts http://developer.android.com/guide/topics/ui/notifiers/toasts.html
- Diálogos http://developer.android.com/guide/topics/ui/dialogs.html