# 29.- Diálogos

Los diálogos son elementos en la interfaz de Android que se sobreponen a las actividades para exigir la confirmación de una acción o solicitar información al usuario.

Estos cuadros de diálogo se caracterizan por interrumpir una tarea del usuario, alterando el foco de la aplicación e invitándolo de forma intrusiva a que vea cierta información, decida ante una circunstancia critica o especifique datos.

En principio, los diálogos de Android los podremos utilizar con distintos fines:

- Mostrar un mensaje.
- Pedir una confirmación rápida.
- Solicitar al usuario una elección (simple o múltiple) entre varias alternativas.

El uso actual de los diálogos en Android se basa en **fragmets**. En este caso nos vamos a basar en la clase `DialogFragment`, para instanciar los dialogo. Aunque es `Dialog` la clase que genera la interfaz, `DialogFragment` es quien permite controlar los eventos de forma fluida. Por ello no se recomienda crear objetos de clase base.

Para crear un diálogo lo primero que haremos será crear una nueva clase que herede de `DialogFragment` y sobrescribiremos uno de sus métodos `onCreateDialog()`, que será el encargado de construir el diálogo con las opciones que necesitemos.

La forma de construir cada díalogo dependerá de la información y funcionalidad que necesitemos.

## Dialogo de Alerta

Este tipo de diálogo se limita a mostrar un **título**, un **mensaje** sencillo al usuario, y un **único botón** de Ok para confirmar su lectura.

En los botones podemos ver acciones como la de aceptación (botón positivo) que determina que el usuario está de acuerdo con la acción. La cancelación (botón negativo) para evitar la realización de una acción, y la neutralidad (botón neutro) para determinar que aún no se está listo para proseguir o cancelar la acción.

Un *dialogo de alerta* se construiye mediante la clase `AlertDialog`, y más concretamente con la su subclase `AlertDialog.Builder`, de forma similar a las notificaciones de barra de estado. Su utilización es muy sencilla, bastará con crear un objeto de tipo `AlertDialog.Builder` y establecer las propiedades del diálogo mediante sus métodos **`setXXX()`**. Siendo los más frecuentes:

- `setTitle()`:Título del diálogo.
- `setMessage()`: Contenido del mensaje que se quiere transmitir.
- `setPositiveButton()`: Crea una instancia del botón de confirmación. El primer parámetro que recibe es el texto del botón y el segundo una escucha `OnClickListener` para determinar que acción se tomará al ser pulsado.

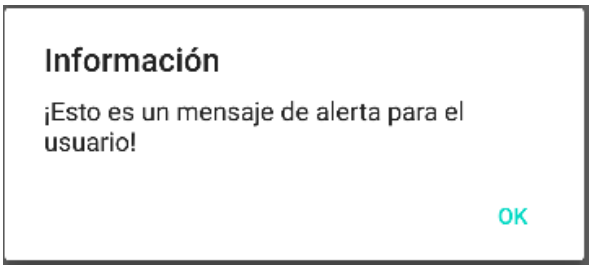
Dentro de este evento, nos limitamos a cerrar el dialogo mediante su método `cancel()`, aunque podríamos realizar cualquier otra acción.

```
package com.example.ejdialogos;

import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AlertDialog;
import androidx.fragment.app.DialogFragment;

public class DialogoAlerta extends DialogFragment {
    @NonNull
    @Override
    public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {

        AlertDialog.Builder builder =
                new AlertDialog.Builder(getActivity());

        builder.setMessage("Esto es un mensaje de alerta para el usuario!")
                .setTitle("Información")
                .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.cancel();
                    }
                });

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

Para lanzar este diálogo, por ejemplo, desde la actividad principal, se obtiene una referencia al *Fragment Manager* mediante una llamada a `getSupportFragmentManager()`, se crea un nuevo objeto de tipo `DialogoAlerta` y por último se muestra el diálogo mediante el método `show()` pasándole la referencia al fragment manager y una etiqueta identificativa del diálogo.

```
FragmentManager fragmentManager= getSupportFragmentManager();
DialogoAlerta dialogoAlerta = new DialogoAlerta();
dialogoAlerta.show(fragmentManager,"tagAlerta");
```

El aspecto de este dialogo de alerta será:



## Dialogo de Confirmación

Un diálogo de confirmación es muy similar al anterior, con la diferencia que se utilizará para solicitar al usuario que confirme una determinada acción, pudiendo incluir hasta 3 botones de confirmación.

En los botones podemos ver acciones como la de aceptación (botón positivo) que determina que el usuario está de acuerdo con la acción. La cancelación (botón negativo) para evitar la realización de una acción, y la neutralidad (botón neutro) para determinar que aún no se está listo para proseguir o cancelar la acción.

La implementación de estos diálogos es prácticamente igual a la comentada para las alertas, salvo que en esta ocasión se añadirán hasta dos botones más

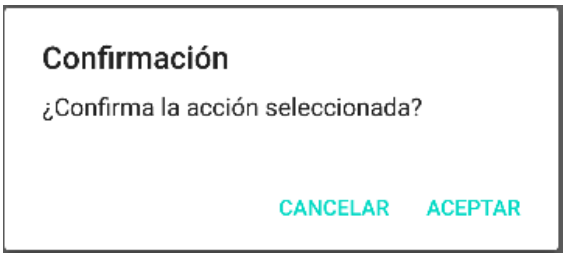
- `setNegativeButton()`:Idem al botón de confirmación pero con la decisión negativa.
- `setNeutralButton()`:Genera la escucha para el botón neutro. Al igual que los dos anteriores recibe el nombre y la escucha.

Veamos un ejemplo:

```
package com.example.ejdialogos;

import android.app.AlertDialog;
import android.app.Dialog;
import android.content.Context;
import android.content.DialogInterface;
import android.os.Bundle;
import android.util.Log;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.DialogFragment;

public class DialogoConfirmacion  extends DialogFragment {

    @NonNull
    @Override
    public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

        builder.setMessage("¿Confirma la acción seleccionada?")
                .setTitle("Confirmación")
                .setPositiveButton("ACEPTAR",
                                   new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        Log.i ("Dialogos", "Confirmación Aceptada");
                        Toast.makeText(getActivity(),
                                "Botón Positivo pulsado", Toast.LENGTH_LONG).show();
                    }
                })
                .setNegativeButton("CANCELAR",
                                   new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        Log.i("Dialogos", "Confirmación denegada");
```

```
                        Toast.makeText(getActivity(),
                                "Botón Negativo pulsado", Toast.LENGTH_LONG).show();
                    }
                });
        return builder.create();
    }

}
```

En este caso, generamos un par de mensajes de log para verificar qué botón del diálogo de confirmación se ha seleccionado. Y su aspecto será:




## Dialogo de Selección

En los diálogos de selección, como su propio nombre indica, el usuario podrá elegir entre una lista de diferentes opciones.

Para ello también se utiliza la clase `AlertDialog`, pero esta vez **no se asigna ningún mensaje** ni se definen las acciones a realizar por cada botón individual, sino que directamente se indica la lista de opciones a mostrar, mediante el método `setItems()` y se proporciona la implementación del evento `onClick()` sobre dicha lista, mediante un listener de tipo `DialogInterface.OnClickListener`, evento en el que se realizan las acciones oportunas según la opción elegida.

La lista de opciones se definiran como un array tradicional.

Aquí se dispone de 3 opciones:

1) El resultado será similar a una **lista**,
2) Silimar a los **RadioButtons** y
3) Similar a **Checkbox**.

En la OPCION 1, el diálogo permite al usuario elegir una opción entre las opciones disponibles que se muestran en pantalla. Pero, ¿y si se quisiéra recordar cuál es la opción u opciones seleccionadas por el usuario para que aparezcan marcadas al visualizar de nuevo el cuadro de diálogo? Para ello se pueden utilizar los métodos `setSingleChoiceItems()` (OPCION 2) o `setMultiChoiceItems()` (OPCION 3), en vez de el `setItems()`.

```
package com.example.ejdialogos;

import android.app.AlertDialog;
```

```
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.util.Log;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.DialogFragment;

public class DialogoSeleccionLista extends DialogFragment {
    @NonNull
    @Override
    public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {

        final String[] items ={"Español", "Ingles", "Frances", "Alemán"};
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

        //OPCION 1:Sólo se puede elegir una opción
        builder.setTitle("Selecciona un idioma:")
                .setItems(items, new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int item) {
                        Log.i ("Dialogos", "Opcion elegida: "+items[item]);
                        Toast.makeText(getActivity(), "Has seleccionado "+
                                items[item], Toast.LENGTH_SHORT).show();
                    }
                });
        return builder.create();
    }
}
```

En la OPCION 2, la forma de utilizarlos es muy similar a la ya comentada. En el caso de `setSingleChoiceItems()`, el método tan sólo se diferencia de `setItems()` en que recibe un segundo parámetro adicional que indica el índice de la opción marcada por defecto. Si no queremos tener ninguna de ellas marcadas inicialmente pasaremos el valor -1.

```
package com.example.ejdialogos;

import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.util.Log;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.DialogFragment;

public class DialogoSeleccionRadioButton extends DialogFragment {

    @NonNull
    @Override
    public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
        final String[] items ={"Español", "Ingles", "Frances", "Alemán"};
```

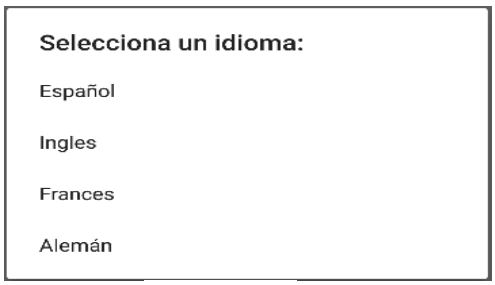
```
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        //OPCION 2: Solo se puede elegir una opción.
        // Se puede sacar una preseleccionada una opción, o indicar
        //con -1 que no se saque ninguna opción preseleccionada.
        builder.setTitle("Selecciona un idioma:")
                .setSingleChoiceItems(items, -1,
                    new DialogInterface.OnClickListener() {
                      @Override
                      public void onClick(DialogInterface dialog, int item) {
                          Log.i ("Dialogos", "Opción Elegida: "+items[item]);
                          Toast.makeText(getActivity(),
                                  "Has seleccionado: "+items[item],
                                  Toast.LENGTH_SHORT).show();
                      }
                })
                .setPositiveButton("Aceptar",
                    new DialogInterface.OnClickListener() {
                      @Override
                      public void onClick(DialogInterface dialog, int which){
                        Toast.makeText(getActivity(),
                                "Opcion Aceptar",Toast.LENGTH_SHORT).show();
                      }
                });
        return builder.create();
    }
}
```

Si por el contrario se opta por la OPCION 3, opción de selección múltiple, la diferencia principal estará en que habrá que implementar un listener del tipo `DialogInterface.OnMultiChoiceClickListener`. En este caso, en el evento `onClick` se recibe tanto la opción seleccionada (`item`) como el estado en el que ha quedado (`isChecked`). Además, en esta ocasión, el segundo parámetro adicional que indica el estado por defecto de las opciones ya no será un simple número entero, sino que tendrá que ser un array de booleanos. En caso de no querer ninguna opción seleccionada por defecto pasaremos el valor `null`.

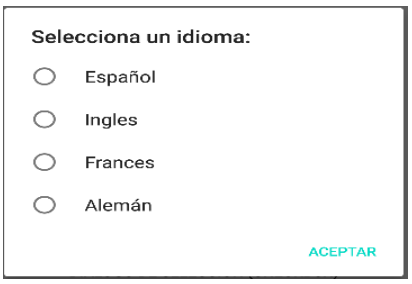
```
package com.example.ejdialogos;

import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.DialogFragment;

import java.util.ArrayList;

public class DialogoSeleccionCheckBox extends DialogFragment {

    @NonNull
    @Override
    public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
        final String[] items ={"Español", "Ingles", "Frances", "Alemán"};
        final ArrayList  itemsSeleccionados = new ArrayList();
        String seleccion;

        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
```

```
            //OPCION 3: Se puede seleccionar más de una opción y
            // puede haber una o más opciones preseleccionadas
            builder.setTitle("Selecciona uno/s idioma/s:")
                    .setMultiChoiceItems(items, null,
                            new DialogInterface.OnMultiChoiceClickListener() {
                                @Override
                                public void onClick(DialogInterface dialog,
                                        int item, boolean isChecked) {
                                    if (isChecked){
                                        itemsSeleccionados.add(item);
                                        Toast.makeText(getActivity(),
                                             "Checks Seleccionados (add): " +
                                              itemsSeleccionados.size(),
                                                Toast.LENGTH_SHORT).show();
                                    }
                                    else if (itemsSeleccionados.contains(item)){
                                        itemsSeleccionados.remove(
                                                  Integer.valueOf(item));
                                        Toast.makeText(getActivity(),
                                              "Checks Seleccionados (remove): " +
                                                     itemsSeleccionados.size(),
                                                Toast.LENGTH_SHORT).show();
                                    }
                                }
                    })
                    .setPositiveButton("Aceptar",
                            new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int item) {
                            Toast.makeText(getActivity(),
                                    "Checks Seleccionados (Aceptar): " +
                                         itemsSeleccionados.size(),
                                    Toast.LENGTH_SHORT).show();
                        }
                    });
            return builder.create();
        }
    }
```

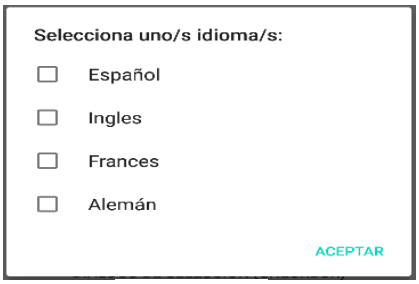
El dialogo en cada uno de los casos quedará de la siguiente manera:




*Opción 1*




*Opción 2*




*Opción 3*

## Dialogo Personalizados

Por último, se puede establecer completamente el aspecto de un cuadro de diálogo. Para ello se actua como si se estuviera definiendo la interfaz de una actividad, es decir, se definirá un layout XML con los elementos a mostrar en el diálogo. Dicho layout se ubicará, como siempre, en la carpeta /res/layout.

Por ejemplo, se define un layout llamado `dialogo_personal.xml` que contendrá una imagen a la izquierda y dos líneas de texto a la derecha:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:src="@drawable/ic_face"
        android:layout_marginTop="20dp"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="20dp"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:padding="3dp"
        android:layout_marginTop="20dp"
        android:layout_marginLeft="20dp">

        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Dialogo_linea_1"
            android:textSize="18dp"/>

        <TextView
            android:id="@+id/textView2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Dialogo_linea_2" />

    </LinearLayout>
</LinearLayout>
```

En el caso de los diálogos personalizados, en el método `onCreateDialog()` correspondiente se utiliza el método `setView()` del builder para asociarle el layout personalizado, que previamente habrá que inflar como ya se ha comentado utilizando el método `inflate()`. Finalmente, se pueden incluir botones como se ha estudiado para los diálogos de alerta o confirmación. En este caso de ejemplo se incluirá un botón de *Aceptar*.

```
package com.example.ejdialogos;

import android.app.AlertDialog;
import android.app.Dialog;
```

```
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.widget.LinearLayout;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.DialogFragment;

public class DialogoPersonalizado extends DialogFragment {

    @NonNull
    @Override
    public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        LayoutInflater inflater = getActivity().getLayoutInflater();
        builder.setView(inflater.inflate(R.layout.dialogo_personal, null))
               .setPositiveButton("Aceptar",
                    new DialogInterface.OnClickListener() {
                      @Override
                      public void onClick(DialogInterface dialog, int which) {
                          dialog.cancel();
                      }
                  });
        return builder.create();
    }
}
```

De esta forma, si se ejecuta de nuevo la aplicación de ejemplo y se lanza el diálogo personalizado se verá algo como lo siguiente:


## Enviar eventos desde un DialogFragment hacia su Actividad Contenedora

Si se quieren pasar los eventos que ocurren en el diálogo hacia una actividad se debe recurrir a una **interfaz de comunicación** que permita compartir los métodos de acción.

Para ello, se deben seguir los siguientes pasos:

**Paso 1**: Declarar una interfaz dentro del `DialogFragment`. La declaración de la interfaz debe tener definido un método para cada acción que reciba el dialogo.

Si se quiere procesar en la actividad los métodos del botón positivo y el negativo, entonces se tiene que crear una interfaz con los dos respectivos métodos:

```
    public interface OnDialogoConfirmacionListener{
        void onPossitiveButtonClick(); //Eventos Botón Positivos
        void onNegativeButtonClick();  //Eventos Botón Negativo
    }
```

**Paso 2**: Declarar un atributo del tipo de la interfaz para conseguir la instancia directa de la actividad

```
// Interfaz de comunicación
OnDialogoConfirmacionListener listener;
```

**Paso 3**: Comprobar que la actividad ha implementado la interfaz, pare ello se puede usar el método `onAttach()`. Recuerda que este método recibe la instancia de la actividad contenedora del fragmento. Simplemente asigna la actividad a la instancia de la interfaz.

Si este proceso no es posible, entonces lanza una excepción del tipo `ClassCastException`.

```
@Override
public void onAttach(Context context) {
    super.onAttach(context);

    try{
        listener = (OnDialogoConfirmacionListener) context;
    } catch (ClassCastException e) {
        throw new ClassCastException(context.toString() +
                " no Implemento OnDialogoConfirmacionListener");
    }

}
```

**Paso 4**: Invocar los métodos de la interfaz en las partes del dialogo donde se desea implicar a la actividad.

```
public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
    AlertDialog.Builder builder =
            new AlertDialog.Builder(getActivity());

    builder.setMessage("¿Confirma la accion selecionada?")
            .setTitle("Confirmacion")
            .setPositiveButton("ACEPTAR",
                    new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    listener.onPossitiveButtonClick();
                }
            })
            .setNegativeButton("CANCELAR",
                    new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    listener.onNegativeButtonClick();
                }
            });
```

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

**Paso 5**: Implementar sobre la actividad contenedora, la interfaz declarada en el fragmento. Y sobreescribir dentro de la actividad los métodos de la interfaz, con las acciones que se requieran

```
public class MainActivity extends AppCompatActivity
                implements DialogoConfirmacion.OnDialogoConfirmacionListener {

    . . .

    @Override
    public void onPossitiveButtonClick() {
        //Acciones a llevar a cabo cuando se pulsa el botón en positivo
        Toast.makeText(this, "Boton Positivo pulsado",
                       Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onNegativeButtonClick() {
        //Acciones a llevar a cabo cuando se pulsa el botón en Negativo
        Toast.makeText(this, "Botón Negativo pulsado",
                       Toast.LENGTH_SHORT).show();
    }
}
```

## ¿Cómo Comunicar Un DialogFragment Con Un Fragment?

Una de las formas para comunicar un fragmento de diálogo y un fragmento común, es tomar la actividad contenedora como puente entre ambos. Es decir, enviar los eventos desde el `DialogFragment` hacia la actividad y luego desde la actividad hacia el `Fragment`.

Esta convención es recomendada, ya que en la documentación oficial de Android Developers se menciona que no se deben comunicar dos fragmentos directamente. Quizás porque las esperanzas de vida de los fragmentos pueden variar, así que es mejor asegurar su independencia de transmisión.

***Veamos este ejemplo para entender mejor…***

Supón que tienes un fragmento sencillo con un `TextView` y un `Button`. La idea es que al presionar el botón, se despliegue un diálogo de lista simple.

Y una vez seleccionada la opción, se actualizará el texto a través de la actividad.

Lo primero es crear un `DialogFragment` que tenga tres opciones. Para este ejemplo usaremos el mensaje "*¿Qué fruta te gusta más?*" y añadiremos tres opciones: *"Naranja", "Mango" y "Plátano"*:

```
package com.example.dialogos22_23;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;

import androidx.annotation.NonNull;
import androidx.fragment.app.DialogFragment;

public class DialogoSeleccionListaFrutas extends DialogFragment {
     OnSetTitleListener listener;


    public AlertDialog crearDialogoSimpleLista (){
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

        final CharSequence[] items = new CharSequence[3];
        items[0]="Naranja";
        items[1]="Mango";
        items[2]="Plátano";

        builder.setTitle("¿Qúé fruta prefieres?")
                .setItems(items, new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialogInterface, int
i) {
                        listener.setTitle((String) items[i]);
                    }
                });
        return builder.create();
    }

    @Override
    public void onAttach(@NonNull Activity activity) {
        super.onAttach(activity);
        try {
            listener = (OnSetTitleListener) activity;
        }catch (ClassCastException e){
            throw new ClassCastException(activity.toString()+"no implementó
OnSetTitleListener");
        }
    }

    //Interface
    public interface OnSetTitleListener {
        void setTitle(String titulo);
    }

}
```

Como ves, la escucha de comunicación `onSetTitleListener` provee un método llamado `setTitle()` para gestionar la selección del usuario. Justo en el momento que el usuario presiona (método `onClick()`), se envía hacia la actividad la cadena contenida en `items[i]`.

Ahora en la actividad principal, se implementará la interfaz y se sobrescribirá el método de implementación:

```
package com.example.dialogos22_23;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.FragmentManager;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity
                    implements DialogoSeleccionListaFru-
tas.OnSetTitleListener {

     //Dentro de una actividad

    @Override
    public void setTitle(String titulo) {
        FragmentoObjetivo fragmentObjetivo
=getSupportFragmentManager().findFragmentByTag("FragmentoObjetivo");
        if (fragmentObjetivo!= null){
            fragmentObjetivo.setTitle(titulo);
        }else{
            //Dentro de una actividad
            Toast.makeText(getApplicationContext(),"No se ha podido estable-
cer el titulo", Toast.LENGTH_LONG).show();
        }

    }
}
```

Al recibir el evento, se debe llamar a la instancia del fragmento al que necesitas pasar los resultados. Para ello se llama al administrador de fragmentos y se usa el método `findFragmentById()` o `findFragmentByTag()`.

El primero obtiene la instancia del fragmento con el id pasado como parámetro y el segundo obtiene el fragmento que contiene la etiqueta especificada.

Con eso ya se puede llamar el método hipotético `setTitle()` del fragmento y comunicar la fruta preferida del usuario en el `Textview`.

# Actividad Propuesta

Crear una aplicación libre, utilizando todo lo visto hasta ahora (Controles básicos, controles de selección, pestañas, Toolbar, menus, temas, estilos,…), en la que antes de empezar se pida autentificación al usuario (usuario: usuario1 y password:123456) y a la hora de finalizar también pida confirmación para terminar la aplicación.