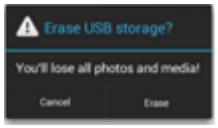
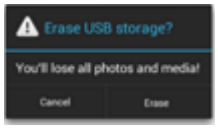
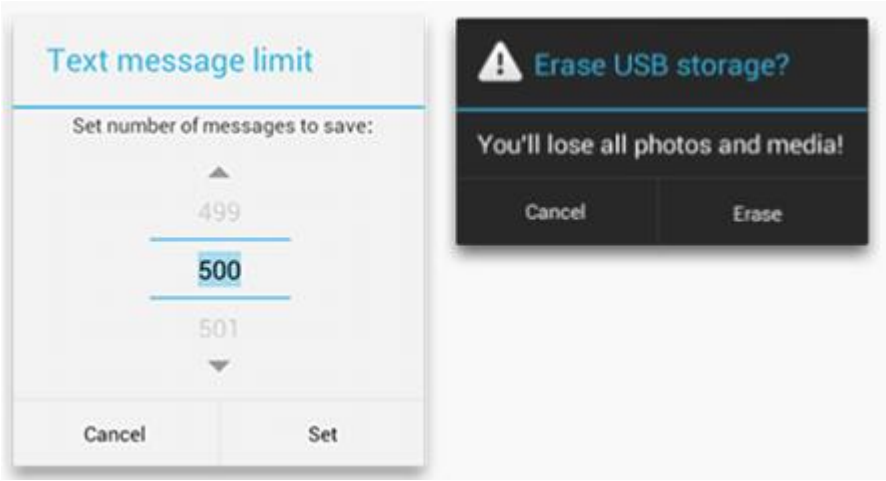

Erase USB storage?
You'll lose all photos and media!
Cancel
Erase

Diálogos

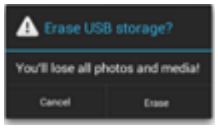
Diálogos



- Un diálogo es una pequeña ventana que pide al usuario que tome una decisión o introducir información adicional.
- Un diálogo no ocupa toda la pantalla.

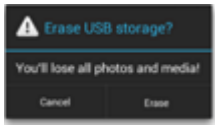


Seleccionar la implementación



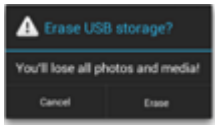
- La plataforma Android crece y cambia rápidamente, sobre todo para soportar nuevos tipos de dispositivos.
- Una de las revisiones más importantes fue la incorporación de los fragmentos y sus ramificaciones para el diseño de la UI.
- Una de estas áreas del diseño es la forma de implementar los diálogos.
- Hay dos maneras de trabajar con diálogos:
  - la clásica, basada en las API's anteriores
  - la recomendada, basada en los fragmentos

Método clásico



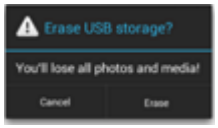
- La clase Activity gestiona los diálogos. Los diálogos se crean, inician y destruyen con métodos de la clase Activity.
- Para el mantenimiento de aplicaciones viejas debería conocerse.
- Está ampliamente documentado con ejemplos
- Android lo marca como obsoleto.
- Pueden tener dificultades en actualizaciones futuras con nuevas funcionalidades del SDK.

Método recomendado



- Los diálogos son un tipo especial de fragmento y se gestionan utilizando la clase FragmentManager.
- Es compatible con las últimas versiones y para poder utilizarlo en las antiguas hay que utilizar el paquete de compatibilidad.
- Es la mejor opción para poder progresar al ritmo de la plataforma.

Ejemplo



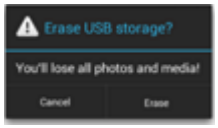
- En la actividad que deba lanzar el diálogo:

  DialogFragment nuevoFragmento = new MiDialogo();

  nuevoFragmento.**show**(getSupportFragmentManager(), "dialogo");

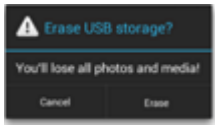
  El segundo argumento, "dialogo", es un nombre de etiqueta único que el sistema usa para guardar y restaurar el estado del fragmento cuando es necesario.
- Y deberemos crear una clase llamada MiDialogo que herede de DialogFragment. En dicha clase fijamos su ciclo de vida utilizando los métodos ya vistos para Fragment. (Aunque no es obligatorio crear la clase aparte porque en la propia actividad podría incluirla, es recomendable porque así dicha clase puede también usarse en otras actividades de la aplicación).

DialogFragment



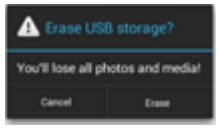
- La clase DialogFragment es la clase base para los diálogos, pero se debe evitar crear instancias directamente. En su lugar, hay que utilizar una de las siguientes subclases:
  - **AlertDialog**: Un cuadro de diálogo que puede mostrar un título, hasta tres botones, una lista de elementos seleccionables o un diseño personalizado.
  - **DatePickerDialog** o **TimePickerDialog**: Un cuadro de diálogo con un interfaz de usuario predefinida que permite al usuario seleccionar una fecha u hora.
- Todos usan DialogFragment como contenedor del diálogo:
  - El uso de DialogFragment para gestionar el diálogo asegura que se controla correctamente los eventos del ciclo de vida, tales como cuando el usuario presiona el botón Volver o gira la pantalla.
  - La clase DialogFragment también permite reutilizar la interfaz de usuario del cuadro de diálogo como un componente integrable en una interfaz de usuario más grande, al igual que una clase Fragment

AlertDialog



- En su construcción debe usarse un elemento auxiliar llamado **Builder**, el cual ayudará a definir las partes del diálogo con gran facilidad y sus eventos de respuesta.
- Builder permitirá fabricar las características del diálogo a través de métodos set*(). Los más frecuentes son:
  - setTitle(): Asigna una cadena al título del diálogo.
  - setMessage(): Asigna el mensaje que deseas transmitir en el contenido.
  - setPossitiveButton(): 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 qué acción se tomará al ser presionado.
  - …
- Después de fijar las características, se materializa el diálogo a través del método create().

Creación de un AlertDialog básico

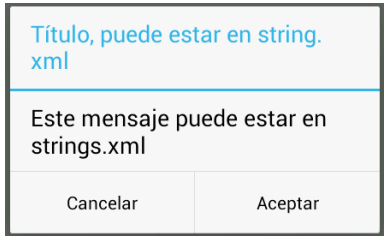



```
public class MiDialogo extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setTitle("Título, puede estar en string.xml")
            .setMessage("Este mensaje puede estar en strings.xml")
            .setPositiveButton(android.R.string.ok,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        Toast.makeText(getActivity(), "Has pulsado Ok", Toast.LENGTH_LONG).show();
                    }
            })
            .setNegativeButton(android.R.string.cancel,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        Toast.makeText(getActivity(),"Has pulsado Cancel", Toast.LENGTH_LONG).show();
                    }
            });
        return builder.create();
    }
}
```

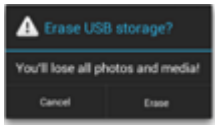
Instanciar un AlertDialog.Builder para el constructor del diálogo

Encadenar varios métodos set para configurar las características de diálogo

Devolver el constructor creado

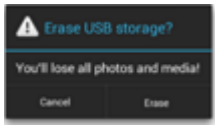


Botones

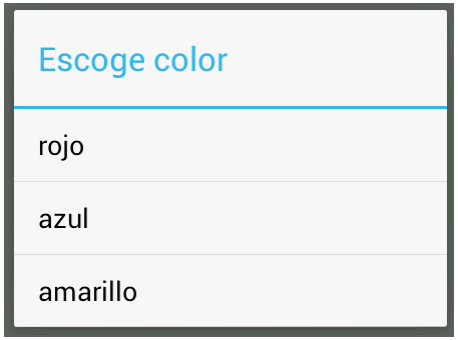


- Un AlertDialog puede tener hasta tres botones.
- Hay tres botones de acción diferentes que se pueden agregar:
  - Positivo : para aceptar y continuar con la acción (el "OK").
  - Negativo: para cancelar la acción.
  - Neutral: cuando el usuario no desee continuar con la acción, pero no necesariamente desea cancelar. Aparece entre los botones positivos y negativos. Por ejemplo, la acción podría ser "Recordármelo más tarde".
- Pueden añadirse solo uno de cada tipo. Es decir, no se puede tener más de un botón "positivo".
- Se añaden con set……Button y requieren un título para el botón (suministrado por un recurso string) y un DialogInterface.OnClickListener que define la acción a realizar cuando el usuario pulsa el botón.

Lista simple

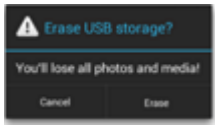



```
public class MiDialogo extends DialogFragment {
        String[] colores = { "rojo", "azul", "amarillo" };
        @Override
        public Dialog onCreateDialog(Bundle savedInstanceState) {
                AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
                builder.setTitle("Escoge color")
                        .setItems(colores, new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog, int which) {
                                        Toast.makeText(getActivity(), "Has escogido "+colores[which], Toast.LENGTH_LONG).show();
                                }
                        });
                return builder.create();
        }
}
```

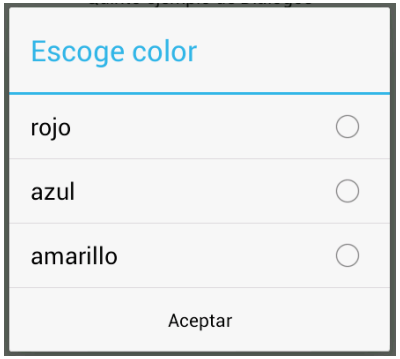


- Obviamente, la lista de opciones puede estar en el fichero strings.xml.
- También puede utilizarse setAdapter() para tener listas dinámicas usando ListAdapter()

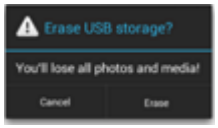
Lista de opciones



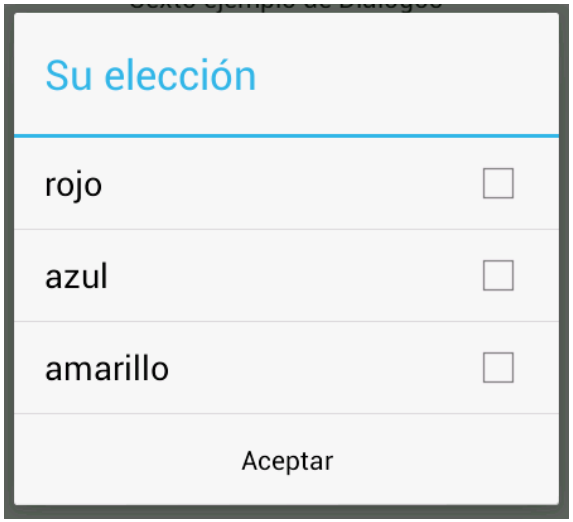

```
public class MiDialogo extends DialogFragment {
        String[] colores = { "rojo", "azul", "amarillo" };
        @Override
        public Dialog onCreateDialog(Bundle savedInstanceState) {
                AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
                builder.setTitle("Escoge color")
                      . setSingleChoiceItems (colores, -1 ,new DialogInterface.OnClickListener() {
                              public void onClick(DialogInterface dialog, int which) {
                                      Toast.makeText(getActivity(), "Has escogido "+colores[which], Toast.LENGTH_LONG).show();
                              }
                      })
                      .setPositiveButton(android.R.string.ok, new DialogInterface.OnClickListener() {
                              public void onClick(DialogInterface dialog, int id) {
                              }
                 });
                return builder.create();
        }
}
```



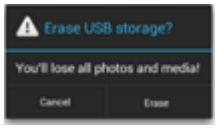
Lista de opciones múltiples




```
public class MiDialogo extends DialogFragment {
    String[] colores = { "rojo", "azul", "amarillo" };
    private Vector<String> items_selecc;
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        items_selecc = new Vector<String>();
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setTitle("Su elección")
            .setMultiChoiceItems(colores, null,
                new DialogInterface.OnMultiChoiceClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which, boolean isChecked) {
                        if (isChecked) {
                            items_selecc.add(colores[which]);
                        } else if (items_selecc.contains(which)) {
                            items_selecc.remove(colores[which]);
                        }
                    }
            })
            .setPositiveButton(android.R.string.ok,
                new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int id) {
                    String resultado = "";
                    Iterator<String> it = items_selecc.iterator();
                    while (it.hasNext()) {
                        resultado = resultado + it.next() + " ";
                    }
                    Toast.makeText(getActivity(), "Has escogido " + resultado,Toast.LENGTH_LONG).show();
                }
        });
        return builder.create();
    }
}
```



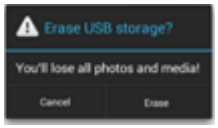
TimerPickerDialog




```
public  class MiDialogo extends DialogFragment implementsTimePickerDialog.OnTimeSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        return new TimePickerDialog(getActivity(), this, hour, minute,DateFormat.is24HourFormat(getActivity()));
    }

    public void onTimeSet(TimePicker view, int hora, int min) {
        Toast.makeText(getActivity(),"Has escogido " + hora+":"+min,Toast.LENGTH_LONG).show();
    }
}
```

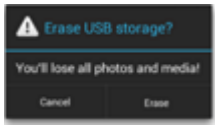
(is24HourView: Determina si el time picker tendrá un formato de 24 horas o AM/PM. Para esto se obtiene las preferencias del usuario para el tiempo con el método is24HourFormat() de la clase DateFormat.)

DatePickerDialog




```
public class MiDialogo extends DialogFragment implements DatePickerDialog.OnDateSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);

        return new DatePickerDialog(getActivity(), this, year, month, day);
    }

    public void onDateSet(DatePicker view, int year, int month, int day) {
        Toast.makeText(getActivity(),"Has escogido " + day + "-" + month + "-" +
        year,Toast.LENGTH_LONG).show();
    }
}
```

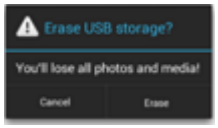
Devolver eventos a la actividad "anfitriona"




```
public class MiDialogo extends DialogFragment {
        public interface MiDialogoListener {
                public void onDialogPositiveClick(String devuelto);
                public void onDialogNegativeClick(int escogido);
        }

        MiDialogoListener miEscuchador;

        // Sobreescribimos el método onAttach() para instanciar el
        //escuchador
        @Override
        public void onAttach(Context context) {
                super.onAttach(context);
                try {
                     miEscuchador = (MiDialogoListener) getActivity();
                } catch (ClassCastException e) {
                throw new ClassCastException(getActivity().toString()
                     + " must implement MiDialogoListener");
                }
        }
```

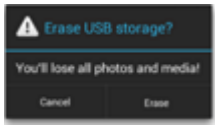
```
        @Override
        public Dialog onCreateDialog(Bundle savedInstanceState) {
                AlertDialog.Builder builder = new
                     AlertDialog.Builder(getActivity());
                builder.setTitle(R.string.app_name)
                .setMessage(R.string.texto4)
                .setPositiveButton(android.R.string.ok,
                new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                     miEscuchador.onDialogPositiveClick("SI");
                }
                })
                .setNegativeButton(android.R.string.cancel,
                new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                     miEscuchador.onDialogNegativeClick(99);
                }
                });
                return builder.create();
        }
        @Override
         public void onDetach () {
              super.onDetach();
              miEscuchador=null;
         }
}
```

Recoger eventos en la actividad "anfitriona"




```
public class MainActivity extends Activity implements OnClickListener,
MiDialogo.MiDialogoListener {
        ……..
        @Override
        public void onDialogPositiveClick(String devuelto) {
                …..
        }

        @Override
        public void onDialogNegativeClick(int escogido) {
                …..
        }
        ……
}
```

Pasar datos desde la actividad "anfitriona"



public class MainActivity extends Activity …

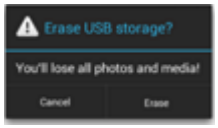
…

newFragment = MiDialogo.*newInstance("le paso esto");*

newFragment.show(getSupportFragmentManager(), "dialogo");

…

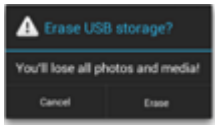
Recoger datos en el diálogo




```
public class MiDialogo extends DialogFragment {

    public static MiDialogo newInstance(String datoRecogido) {
        MiDialogo newInstance = new MiDialogo();
        Bundle args = new Bundle();
        args.putString("dato", datoRecogido);
        newInstance.setArguments(args);
        return newInstance;
    }
    …
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        final String tipo = getArguments().getString("dato");
    …
}
```

- Naturalmente, podemos pasar más de un dato y del tipo que se necesite

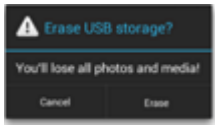
Diálogos Personalizados



- Podemos establecer completamente el aspecto de un cuadro de diálogo definiendo un layout XML con los elementos a mostrar en el diálogo.
- En el método onCreateDialog() correspondiente utilizaremos el método setView() del builder para asociarle nuestro layout personalizado, que previamente tendremos que inflar utilizando el método inflate() y fijaremos los "listener" necesarios.

```
public class DialogoPersonalizado extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        LayoutInflater inflater = getActivity().getLayoutInflater();
        View view= inflater.inflate(R.layout.dialog_personal, null);
        builder.setView(view);
        Button qqq= (Button) view.findViewById(R.id.button2);
        qqq.setOnClickListener(
            new View.OnClickListener() {
              @Override
              public void onClick(View v) {
                // Acciones a realizar...
                dismiss();
              }
            }
        );
        return builder.create();
    }
}
```

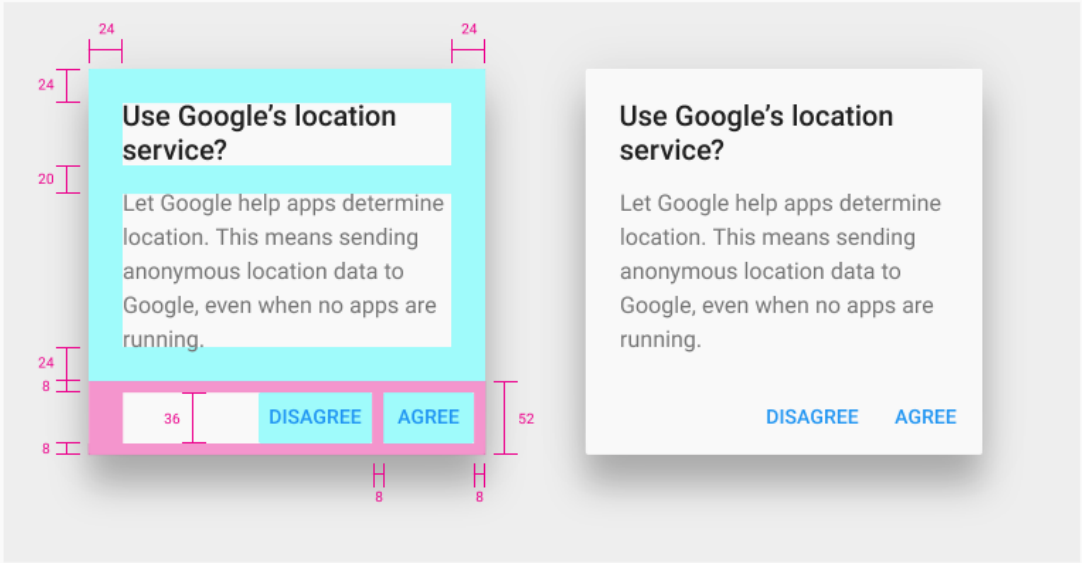
Diseño de layout para diálogos personalizados



- Deben seguirse las [normas de diseño recomendadas] por Material Design.

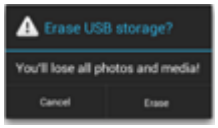
Content guidelines

Padding around content area: 24dp
Padding between title and body text: 20dp
Padding around buttons: 8dp
Button height: 36dp
Action area height: 52dp
Dialog elevation: 24dp



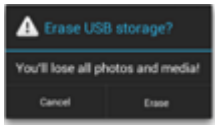
Content padding

Diálogos en Fragmentos



- 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.

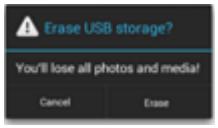
Descartar un diálogo



- Afortunadamente los diálogos son descartados en el momento que sus botones de acción son presionados o incluso si el usuario presiona fuera de su contenido. Pero en ocasiones muy extremas tal vez se requiera cerrar el diálogo manualmente.
- Y esto es tan sencillo como usar el método dismiss() dentro de DialogFragment.
- Adicionalmente pueden realizarse acciones en ese instante con el método onDismiss():

```
// Dentro de un DialogFragment...
@Override
public void onDismiss(DialogInterface dialog) {
    // acciones
}
```

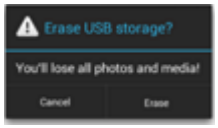
Cancelar un diálogo



- El diálogo puede ser cancelado sin aplicar los cambios a través del método cancel().
- Para procesar su comportamiento puede usarse onCancel(), el cual es invocado si el usuario presiona el Back Button o si se presiona fuera del diálogo:

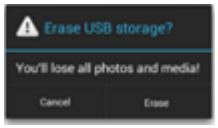
```
// Dentro de un DialogFragment...
@Override
public void onCancel(DialogInterface dialog) {
    // acciones
}
```

Mostrar un diálogo en pantalla completa



- El uso de diálogos en pantalla completa se puede dar cuando el diálogo contiene varios elementos agrupados que requieren más espacio.
- Lo curioso es que al crear un diálogo en pantalla completa, no podemos usar la clase AlertDialog junto a Builder, debido a que el DialogFragment será embebido a través de una transacción de fragmentos como se hace normalmente.
- Esto significa que el método onCreateDialog() no será sobrescrito, por lo que debemos acudir a onCreateView() para inflar un layout por completo.
- Así que puedes deducir que las áreas de título, contenido y acción no están distribuidas de la misma forma. Esta vez los botones de confirmación y el título van en la action bar y el contenido será ubicado en todo el espacio de trabajo.
- Más información

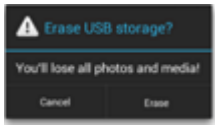
Mostrar actividad como si fuese un diálogo



- En lugar de mostrar un diálogo como una IU de pantalla completa, se puede lograr el mismo resultado mostrando una Activity como un diálogo (y es mucho menos laborioso! ☺).
- Basta con fijar a dicha actividad un tema con dicha característica:

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

Prácticas propuestas



- Realiza la hoja de Ejercicios_13_Dialogos