

U.T.4 PROGRAMACIÓN PARA DISPOSITIVOS MÓVILES. INTENT



MENSAJE DE TEXTO (*Toast*)

Son pequeños textos que se muestran al usuario de forma breve, como flotando sobre la ventana principal, y desaparecen automáticamente un instante después.

El método `makeText` toma tres parámetros: El contexto, el mensaje, y la duración de visualización del éste y devuelve un objeto `Toast` que se lanza con `show()`.

```
Context contexto = getApplicationContext();
int duracion = Toast.LENGTH_SHORT; // Toast.LENGTH_LONG
Toast toast = Toast.makeText(contexto, R.string.<recurso>, duracion);
toast.show();
```

El ejemplo anterior muestra un mensaje en la parte inferior de la ventana. Podemos personalizarlo cambiando su posición en la pantalla con el método `setGravity()`

```
toast.setGravity(Gravity.CENTER|Gravity.LEFT,0,0);
```

ALERTAS O CUADROS DE DIÁLOGOS. AlertDialog.

Los cuadros de diálogos en Android los podemos utilizar con distintos fines:

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

Actualmente los cuadros de diálogos en Android están basados en [Fragment](#), esto nos permite personalizarlos. Para crear un diálogo lo primero que haremos será crear una nueva clase que herede de [DialogFragment](#) y sobrescribir el método [onCreateDialog\(\)](#), que será el encargado de construir el diálogo con las opciones que necesitemos.

Diálogo de Alerta

Muestra un mensaje sencillo al usuario, y un único botón de OK para confirmar su lectura. Lo construiremos mediante la clase [AlertDialog](#), y más concretamente su subclase [AlertDialog.Builder](#).

Para establecer las propiedades del diálogo usamos sus métodos:

[setTitle\(\)](#) para el título.

[setMessage\(\)](#) para el mensaje.

[setPositiveButton\(\)](#) para el texto y comportamiento del botón.

```
public class Dialogo extends DialogFragment {  
    @Override  
    public Dialog onCreateDialog(Bundle savedInstanceState) {  
        AlertDialog.Builder alerta =new AlertDialog.Builder  
            (getActivity());  
        alerta.setMessage(R.string.aler1);  
        alerta.setTitle(R.string.tit1);  
        alerta.setPositiveButton(android.R.string.ok, new  
            DialogInterface.OnClickListener() {  
                public void onClick ( DialogInterface dialog, int id)  
                {  
                    dialog.cancel(); }  
            });  
        return alerta.create();  
    }  
}
```

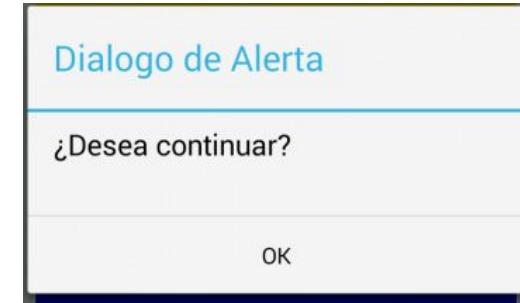
Lanzar el cuadro de diálogo

Para lanzar el cuadro de diálogo desde nuestra actividad principal, necesitamos una referencia a `FragmentManager` mediante una llamada al método `getSupportFragmentManager()` o `getFragmentManager()`, instanciamos la clase `Dialogo` creada anteriormente y mostramos el cuadro de diálogo con `show()`. pasándole la referencia a `fragmentManager` y una etiqueta identificativa del diálogo.

```
FragmentManager fragmentManager = getFragmentManager();
```

```
Dialogo dialogo = new Dialogo();
```

```
dialogo.show(fragmentManager, "tagDialogo");
```

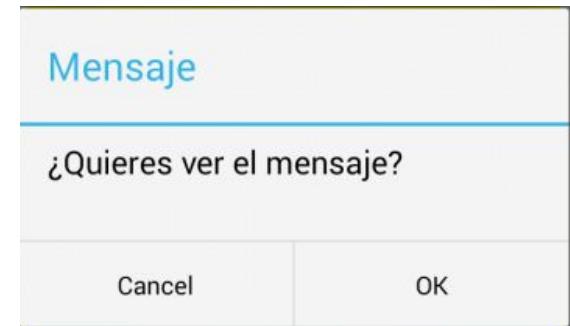


Diálogo de Confirmación

Un diálogo de confirmación es muy similar al anterior, con la diferencia de que lo utilizaremos para solicitar al usuario que nos confirme una determinada acción, por lo que las posibles respuestas serán del tipo Sí/No.

Para la implementación añadiremos dos botones, uno de ellos para la respuesta afirmativa `setPositiveButton()`, y el segundo para la respuesta negativa `setNegativeButton()`. Puede haber un tercer botón `setNeutralButton()` por ejemplo para cancelar.

Para establecer un icono usamos `setIcon(<id>)` . Existen iconos predefinidos en Android.



Intent

Es una herramienta para realizar invocaciones a otros componentes de la aplicación. Los casos de uso fundamentales son:

- Para invocar una actividad (**Activity**).
- Para invocar un servicio. (**Service**).
- Para entregar una emisión, mensaje del sistema. (**Broadcast**).

Hay dos tipos de **intent**:

- **explícitos**: Debe especificar el nombre del componente que se quiere invocar. Por ejemplo el nombre de una actividad o servicio que se desea iniciar.
- **implícitos**: No nombran un componente específico, pero en su lugar declara una acción que debe ser llevada a cabo.

INVOCAR A OTRA ACTIVIDAD.

Un **intent** es una descripción abstracta de una operación que queremos que se realice, por lo que puede verse como un mensaje asíncrono. Es recogido por Android, que lo procesa y **lanza el componente (activity)** que se haya pedido.

En este ejemplo, vamos a hacer uso de lo que se conoce como **modo de invocación explícito**: hemos configurado con total exactitud la actividad que queremos que se lance, dando el nombre de su clase Java.

```
Intent i = new Intent (this, nombreActividad.class);
```

```
startActivity(i); //inicia la actividad indicada en el intent
```

PASO DE DATOS ENTRE ACTIVIDADES.

Cuando configuramos un **intent**, podemos añadir información (**lista de pares atributo-valor**) que podrá ser leída por la actividad invocada.

- Desde el **lado del invocante**, la clase **Intent** usa el método **putExtra(atributo, valor)** para almacenar los datos que desea pasar a la otra actividad. En este caso nombre y edad.

```
Intent i = new Intent (this, actividadInvocada.class);
```

```
i.putExtra("nombre", "Ana");
```

```
i.putExtra("edad", 25);
```

```
startActivity(i);
```

- Desde el **punto de vista del invocado**, la actividad tiene acceso al **intent** a través del método **getIntent()** y puede conseguir los valores con los métodos **getStringExtra()** para nombre y **getIntExtra()** para edad.

```
String name = getIntent().getStringExtra("nombre");
```

```
int edad = getIntent().getIntExtra("edad");
```

ENVÍO RECÍPROCO DE DATOS ENTRE DOS ACTIVIDADES

Al igual que desde la actividad invocante podemos enviar datos a la otra actividad, también podemos recibir datos de la actividad invocada.

Desde el lado del invocante, al ejecutar el intent, en vez de `startActivity()` usaremos el método `startActivityForResult()` y recogeremos de manera asíncrona el resultado en su método `onActivityResult()`.

```
Intent i = new Intent (this, actividadInvocada.class);
i.putExtra("emisor", "Ana");
startActivityForResult(i, codigoRespuesta);
protected void onActivityResult(int requestCode,
int resultCode, Intent data)
{
    String respuesta;
    receptor = data.getStringExtra("receptor");
    ...
}
```

ENVÍO RECÍPROCO DE DATOS ENTRE DOS ACTIVIDADES

Desde el **punto de vista del invocado**, La segunda ventana creará un nuevo intent que enviará de vuelta a la actividad invocante usando el método:

setResult(RequestCode, intent)

Donde **requestCode** puede tomar los valores :

RESULT_CANCELED

Constant Value: 0

RESULT_FIRST_USER

Constant Value: 1

RESULT_OK

Constant Value: -1

```
Intent ii = new Intent();
ii.putExtra("receptor", "Jaime");
setResult(RESULT_OK, ii);
```

INVOCAR UNA ACTIVIDAD DE OTRA APLICACIÓN

Para lanzar una actividad (ventana) de otra aplicación debemos crear el intent e indicarle el nombre de la aplicación y el nombre completo de la actividad que queremos lanzar llamando al método `setClassName` como sigue:

```
Intent i = new Intent();
i.setClassName("dam.moviles.dosactivities","dam.moviles.dosactivities.
SegundaActividad");
```

La actividad que queremos lanzar debe aparecer en el archivo `Android Manifest` como principal, si no añade un `intent-filter` que dé permiso para la ejecución externa, como sigue:

```
<activity
    android:name=".SegundaActividad"
    android:label="@string/title_segunda_ventana" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
    </intent-filter>
</activity>
```

INVOCACIONES IMPLÍCITAS.

Los `intent` se han diseñado para usarlos de una manera mucho más potente llamada **uso implícito**, en el que **no se proporciona qué actividad debe procesar la solicitud** y en su defecto se proporcionará una serie de campos para deducir qué actividad es capaz de atender la solicitud. Que son:

- **Acción:** es una cadena que indica la acción que se desea realizar. Puede ser `ACTION_VIEW`, `ACTION_EDIT`, `ACTION_DIAL`,... Se establece con `setAction()`, o incluso pasándola en el constructor. Es la acción la que determina cómo se utilizará el resto de la información contenida en el intent.
- **Datos:** una URI sobre la que ejecutar la acción. Se establece con el método `setData(Uri u)`. Si URI se proporciona como String será necesario convertirla con `setData(Uri.parse("..."))`.
- **Tipo de los datos:** si a partir de la Uri no es posible identificar el tipo de datos, podemos indicarlo explícitamente al intent con `setType(String type)`. Donde type indica el tipo **MIME** (por ejemplo `image/*`).

Podemos establecer la URI y el tipo simultáneamente con `setDataAndType(Uri, String)`. En ocasiones, queremos especificar un tipo de datos pero no una URI.

- **Categorías** Una cadena que contiene información adicional sobre el tipo de componente que debe manejar la intención. La mayoría de intent no requieren categoría. Ej. `CATEGORY_BROWSABLE`; `CATEGORY_LAUNCHER`,..

INVOCACIONES QUE IMPLICAN ACCIÓN

- Lanzar la actividad que muestra el resumen de uso y estado de la batería.

```
Intent i = new Intent();
i.setAction(Intent.ACTION_POWER_USAGE_SUMMARY);
startActivity(i);
```

- Lanzar una página web.

```
Intent i = new Intent();
i.setAction(Intent.ACTION_VIEW);
i.setData(Uri.parse("http://www.google.es"));
startActivity(i);
```

- Lanzar una llamada

```
Intent i = new Intent();
i.setAction(Intent.ACTION_DIAL);
i.setData(Uri.parse("tel:5554433"));
```

PERMISOS

Para invocar algunas de las acciones anteriores necesitamos establecer los permisos correspondientes en el archivo de Android Manifest.

- El fichero `AndroidManifest.xml`, que podrás encontrar en la raíz del proyecto. Contiene información esencial sobre la aplicación, en este fichero se definen los diferentes elementos que la componen. Por tanto, todas las actividades de nuestra aplicación deben quedar convenientemente recogidas en este fichero. `AndroidManifest.xml` organiza toda la información que contiene en cuatro pestañas diferentes (**Manifest**, **Application**, **Permission** y **Instrumentation**). También muestra una quinta pestaña con la vista del fichero en su versión en XML.
- Para añadir un permiso en `AndroidManifest.xml`

```
<uses-permission  
    android:name="android.permission.CALL_PHONE"/>
```

Para más información de permisos:

<https://developer.android.com/reference/android/Manifest.permission.html>