TEMA 4. ACTIVIDADES E INTENCIONES

1. ACTIVIDADES

1.1. Introducción

Las actividades son el **principal componente de las aplicaciones** en Android, y se encarga de **gestionar gran parte de las interacciones con el usuario**. Corresponde a lo que coloquialmente llamamos una **pantalla de la aplicación**. Una **aplicación** suele estar **formada por una serie de actividades**, de forma que el usuario puede ir navegando entre actividades. En concreto, Android suele disponer de un botón (físico o en pantalla) que nos permite volver a la actividad anterior.

A nivel de lenguaje de programación, una actividad **hereda de la clase Activity** y se traduce en una pantalla. Así, toda actividad ha de tener una vista asociada, que será utilizada como interfaz de usuario. Esta vista suele ser de tipo *layout*, aunque también puede ser una vista simple.

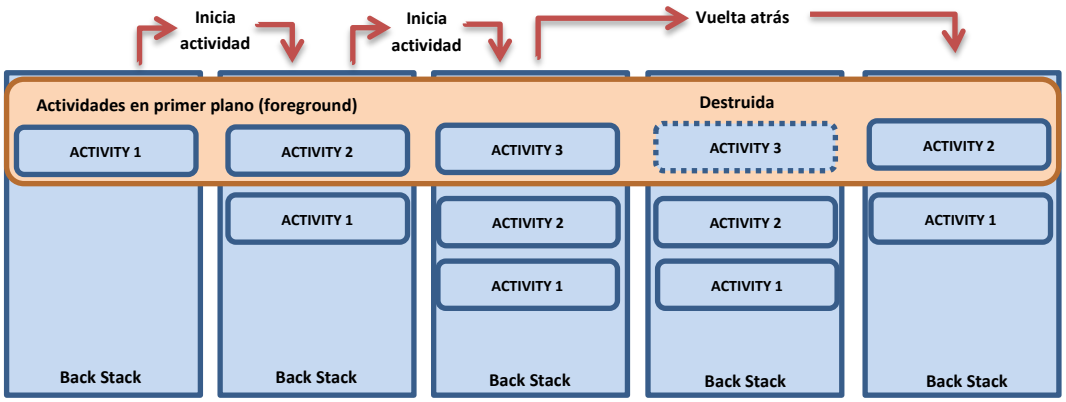
Una aplicación estará formada por un **conjunto de actividades independientes**; es decir, se trata de clases independientes que no comparten variables, aunque todas trabajan para un objetivo común.

Cada vez que vayamos a **crear una nueva actividad** en nuestra aplicación, deberemos seguir los siguientes pasos:

- Crear un nuevo Layout para la actividad.
- Crear una nueva clase descendiente de Activity. En esta clase tendrás que indicar que el Layout a visualizar es el desarrollado en el punto anterior.
- Para que nuestra aplicación sea visible será necesario activarla desde otra actividad.
- De forma obligatoria tendremos que **registrar toda nueva actividad** en *AndroidManifest.xml*.

1.2. Funcionamiento

Por lo general, una aplicación Android consta de múltiples actividades que están más o menos ligadas entre sí. Habitualmente, se define una actividad "principal", que es la que se presenta al usuario cuando se inicia la aplicación por primera vez. Una actividad puede iniciar otra actividad con el fin de realizar diferentes operaciones. Cada vez que comienza una nueva actividad, la actividad anterior se detiene y se envía a una pila de retroceso (*back stack*). Esta pila usa el mecanismo de cola LIFO (*last in, first out*), por lo que, cuando el usuario pulsa la tecla "Volver atrás" del dispositivo, se extrae de la pila la actividad anterior (destruyéndose la pila) y se reanuda.



Cuando una actividad se para porque se inicia una nueva actividad, se le notifica este cambio de estado a través de los **métodos de llamada o *callback*** del ciclo de vida de la actividad.

Un método de llamada (*callback*) es una función que se remite a Android cuando se inicia una Actividad, para que el sistema operativo la "llame" durante la ejecución de esta Actividad.

Existen varios métodos de llamada *callback* que una actividad puede recibir debido a un cambio en su estado; por ejemplo, cuando el sistema crea la Actividad, cuando se reactiva o cuando se destruye. El programador puede aprovechar estos métodos para ejecutar sentencias específicas apropiadas para el cambio de estado. Por ejemplo, cuando una Actividad se suspende es recomendable liberar de la memoria todos los objetos grandes. Cuando la actividad se reanuda, se puede volver a reservar los recursos necesarios y continuar con las acciones que se interrumpieron. Estos cambios de estado forman parte del ciclo de vida de la actividad.

A partir de la versión 3 de Android (HONEYCOMB), una actividad puede usar la clase Fragmento (*Fragment*) para separar en módulos mejor su código, construir interfaces de usuario más sofisticadas en pantallas más grandes y ayudar a escalar la aplicación entre las pantallas grandes y pequeñas.

1.3. Ciclo de vida de una actividad

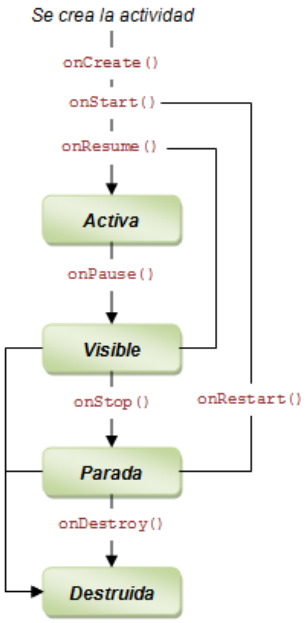
Una actividad puede estar en diferentes periodos de funcionamiento que denominaremos ***estados***, y que van a depender del flujo de la aplicación, de la interacción con el usuario y de las necesidades de recursos del sistema.

La actividad debe relacionarse con el sistema operativo y con otros programas que pueden estar en diferentes estados en el dispositivo móvil.

Una actividad en Android puede estar en uno de estos cuatro estados:

- **Activa (*running o resumed*)**: La actividad está encima de la pila, lo que quiere decir que es visible y tiene el foco.
- **Pausada (*paused*)**: La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada por completo, pasa a estar parada.
- **Parada (*stopped*)**: Cuando la actividad no es visible. El programador debe guardar el estado de la interfaz de usuario, preferencias, etc. Sí ocupa memoria.
- **Destruida (*destroyed*)**: Cuando la actividad termina al invocarse el método finish(), o es matada por el sistema.

Cuando una actividad cambia entre los diferentes estados descritos anteriormente, el sistema operativo le notifica el cambio mediante diferentes métodos callback. El programador puede usar todos estos métodos callback para ejecutar las órdenes apropiadas.



Los métodos callback son los siguientes.

- ***onCreate(Bundle)***: se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de la actividad (en una instancia de la clase Bundle), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.

- ***onStart():*** nos indica que la actividad está a punto de ser mostrada al usuario.
- ***onResume():*** se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.
- ***onPause():*** indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra actividad es lanzada. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.
- ***onStop():*** este método es invocado cuando la actividad ya no es visible al usuario y otra actividad ha pasado a primer plano. Se pausan animaciones, determinados servicios se detienen (GPS) y se minimizan los recursos consumidos por la actividad, aunque la actividad aún está en memoria. Si queremos reactivarla se utiliza *onRestart()*, volviendo a primer plano, u *onDestroy()* si queremos eliminarla definitivamente. ¡Ojo si hay muy poca memoria!, ya que es posible que la actividad se destruya sin llamar a este método.
- ***onRestart():*** llamado cuando una actividad parada vuelve a primer plano. Siempre es seguido de un onStart().
- ***onDestroy():*** Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón de volver o cuando se llama al método finish(). Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

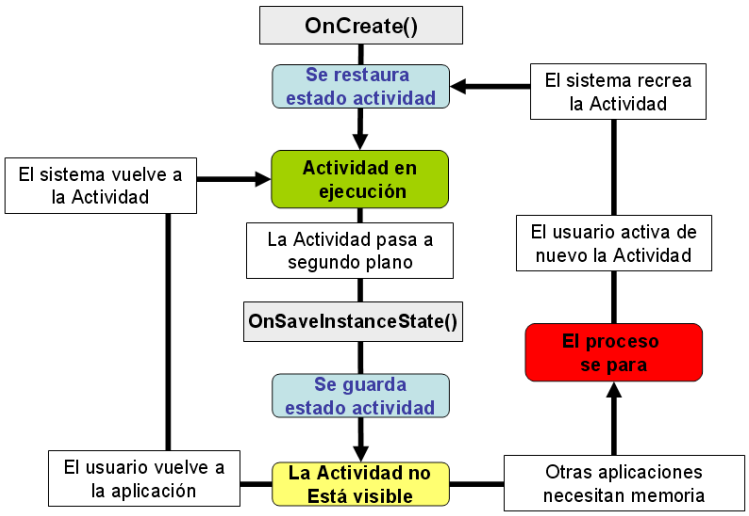
1.4. Guardar y recuperar el estado de una actividad

La configuración de un teléfono puede cambiar mientras un usuario ejecuta una aplicación. Por ejemplo, la orientación de la pantalla (vertical/horizontal) puede alternar, la disponibilidad del teclado, el idioma, etc. Cuando este cambio se produce, Android reinicia la Actividad usando el método *OnDestroy()* e inmediatamente invoca el método *onCreate().*

Este comportamiento de reinicio está diseñado para que la aplicación se adapte a la nueva configuración de forma automática, por ejemplo, cuando cambia la posición de los componentes.

La mejor manera de manejar un cambio de configuración para preservar el estado de la aplicación es utilizar los métodos *onSaveInstanceState()* y *onCreate()* u *onRestoreInstanceState()*.

En el siguiente esquema se muestra los eventos y estados por los que pasa una Actividad cuando se detiene o se destruye.



Como se puede ver en el esquema anterior, una Actividad recupera su estado de ejecución si se detiene y vuelve a primer plano; o si se destruye y se vuelve a crear.

Para almacenar la información importante sobre el estado de ejecución de una Actividad podemos usar el método de llamada de Android *onSaveInstanceState()* y luego restaurar esta información cuando el sistema vuelva a crear la Actividad.

El sistema llama a este método justo antes de que la Actividad se destruya y le pasa como parámetro un objeto de tipo *Bundle*. Este objeto *Bundle* es donde podemos almacenar la información del estado de la Actividad como pareja nombre-valor, utilizando el método *putString()*. De esta manera, si Android mata el proceso de la Actividad y, después, se vuelve a ejecutar la misma Actividad, el sistema pasa el objeto *Bundle* almacenado anteriormente como parámetro en el método *onCreate()* y en el método *onRestoreInstanceState()* (podemos utilizar cualquiera de los dos métodos) para que pueda restaurar el estado de ejecución. Si no hay información sobre este estado, el objeto *Bundle* es nulo.

El esquema gráfico de este comportamiento es el siguiente:

Resumiendo:

- ***onSaveInstanceState(Bundle):*** la invoca el sistema cuando ha de destruir una actividad, que más adelante ha de restaurar (por cambiar su inclinación o por falta de memoria), para permitir a la actividad guardar su estado.

- ***onRestoreInstanceState(Bundle):*** se invoca cuando se restaura la actividad para recuperar el estado guardado por onSaveInstanceState().

Debido a que Android no garantiza que siempre se invoque el método onSaveInstanceState(), debe usarse únicamente para almacenar el estado transitorio de la Actividad, es decir, la interfaz del usuario. Nunca debe emplearse para almacenar datos persistentes como las preferencias del usuario o los datos de la aplicación. Para guardar estos datos utiliza *onPause()* o *onStop()*.

Incluso si no lo hacemos en el método *onSaveInstanceState*(), la aplicación **restaura por defecto automáticamente parte del estado de la Actividad**. En concreto, la implementación por defecto de este método recorre cada componente de la interfaz de usuario, guardando y restaurando automáticamente estos componentes. Por ejemplo, se almacena automáticamente el texto contenido en un componente de tipo EditText o la selección en un componente CheckBox. Lo único que debemos hacer para que Android guarde el estado de los componentes es proporcionar un identificador único (atributo android:id) para cada uno de ellos. Si un componente no tiene un identificador, entonces no se guarda su estado.

También es **posible indicar de forma explícita** que no se almacene automáticamente el estado de un componente de dos formas: en el diseño de la interfaz de usuario con el atributo ***android:saveEnabled*** establecido a "false", o usando el método del componente ***setSaveEnabled***(). Por lo general, no se debe desactivar esta opción, salvo que debamos proteger cierta información (contraseñas) o que al restaurar el estado de la interfaz de usuario haya que mostrar la pantalla de manera diferente.

Aunque el método *onSaveInstanceState*() guarda la información útil del estado de la interfaz de usuario, puede ser necesario guardar información adicional, como la orientación del dispositivo (horizontal o vertical). Podemos usar este método y la variable Bundle para almacenar esta información extra. Ten en cuenta que siempre debemos invocar el método de la clase madre antes de guardar el resto de información:

```
super.onCreate(savedInstanceState);
```

2. INTENTS

2.1. Introducción

Una **intención** representa la voluntad de realizar alguna acción o tarea, como realizar una llamada de teléfono o visualizar una página web. Los *intents* nos permiten lanzar una actividad o servicio de nuestra aplicación o de una aplicación diferente. Formalmente, es un sistema de comunicación que permite interactuar entre componentes de la misma o de distintas aplicaciones de Android.

Existen dos tipos de intenciones, explícitas e implícitas.

2.2. Intents explícitos

En los Intents explícitos se indica exactamente el componente a lanzar. Su utilización típica es la de ir ejecutando los diferentes componentes internos de una aplicación.

Su construcción es fácil, que ya que tan solo se debe instanciar el Intent pasándole como parámetro el contexto y la actividad que se va a lanzar. Posteriormente, se lanzaría mediante el comando startActivity.

```
Intent actividad2 = new Intent(Actividad1.this,Actividad2.class);
startActivity(actividad2);
```

Todas las actividades que se ejecuten de manera independiente deben ser previamente declaradas en el Manifest.

```
<!-- Añadimos la nueva actividad -->
<activity android:name=".Actividad2"></activity>
```

El archivo *manifest* quedaría de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.ejemplointentsexplicitos">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.EjemploIntentsExplicitos"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- Añadimos la nueva actividad -->
        <activity android:name=".Actividad2"></activity>

    </application>
```

2.3. Intents implícitos

Son aquellos con los que tan solo informamos al sistema de la acción que deseamos realizar, y el sistema nos responde con el componente más adecuado para realizar dicha petición, es decir, no se especifica un componente concreto. Así, si se quisiera ver una página web, sería el sistema el que propondría el navegador con el que hacerlo o preguntaría con cuál hacerlo si tuvieras más de uno instalado en el dispositivo y ninguno configurado como predeterminado.

2.4. Elementos

Cuando se crea una Intención (es decir, se instancia un objeto de tipo Intent) esta contiene información de interés para que el sistema trate adecuadamente la intención o para el componente que recibe la intención. Puede incluir la siguiente información:

2.4.1 Nombre del componente

Se utiliza fundamentalmente con los Intents explícitos. Es donde se identifica el componente que se desea lanzar con el Intent.

```
Intent actividad2 = new Intent(Actividad1.this,Actividad2.class);
startActivity(actividad2);
```

2.4.2 Acción

Una cadena de caracteres donde indicamos la acción a realizar y lo identifica dentro del *Manifest*, o en caso de un Receptor de anuncios (Broadcast receiver), la acción que tuvo lugar y que queremos reportar. La clase Intent define una serie de constantes para acciones genéricas que se enumeran a continuación:

Constante	Acción
ACTION_VIEW	Visualiza un contacto, mapa, página Web,… o reproduce música o vídeo,
ACTION_INSERT	Inserta un contacto o cita en calendario
ACTION_EDIT	Edita un contacto o cita en calendario
ACTION_MAIN	Arranca como actividad principal de una tarea
ACTION_CALL	Inicializa una llamada de teléfono
ACTION_DIAL	Introduce un número sin llegar a realizar la llamada
ACTION_SEND	Manda un correo o SMS
ACTION_SET_TIMER	Programa un temporizados
ACTION_SET_ALARM	Programa una alarma.
ACTION_IMAGE_CAPTURE	Tomar una foto
ACTION_VIDEO_CAPTURE	Grabar un vídeo
ACTION_GET_CONTENT	Seleccionar un tipo de archivo específico
ACTION_OPEN_DOCUMENT	Abrir un tipo de archivo específico
ACTION_CREATE_DOCUMENT	Crear un tipo de archivo específico
ACTION_PICK	Seleccionar un contacto o fichero

También puedes definir tus propias acciones. En este caso has de indicar el paquete de tu aplicación como prefijo. Por ejemplo:
org.example.mislugares.MUESTRA_MAPA_LUGARES

Ejemplo: *invocar al sistema para realizar una llamada telefónica*

```
Intent ejemplo = new Intent(Intent.ACTION_CALL);
ejemplo.setData(Uri.parse("tel:NUMERO"));
activity.startActivity(ejemplo);
```

2.4.3 Datos

Referencia a los datos con los que trabajaremos. Contiene la URI (*Uniform Resource Identifier*) con los datos con los que el componente que reciba el Intent debe trabajar. El tipo de datos determina qué componente usará el sistema para procesarlos. Así, por ejemplo, para construir un Intent implícito que le pida al sistema enviar un correo electrónico, debería hacerse de la siguiente forma:

```
Intent correo = new Intent();
correo.setAction(Intent.ACTION_SEND);
correo.putExtra(Intent.EXTRA_TEXT,mensaje);
correo.setType("text/plain");
```

2.4.4 Categoría

Complementa a la acción. Indica información adicional sobre el tipo de componente que ha de ser lanzado. El número de categorías puede ampliarse arbitrariamente. No obstante, en la clase Intent se definen una serie de categorías genéricas que podemos utilizar.

Constante	Significado
CATEGORY_BROWSABLE	La actividad lanzada puede ser invocada con seguridad por el navegador para mostrar los datos referenciados por un enlace (por ejemplo, una imagen o un mensaje de correo electrónico).
CATEGORY_HOME	La actividad muestra la pantalla de inicio, la primera pantalla que ve el usuario cuando el dispositivo está encendido o cuando se presiona la tecla *Home*.
CATEGORY_LAUNCHER	La actividad puede ser la actividad inicial de una tarea y se muestra en el lanzador de aplicaciones de nivel superior.
CATEGORY_PREFERENCE	La actividad a lanzar es un panel de preferencias.

2.4.5 Extras

Información adicional que será recibida por el componente lanzado. Está formada por un conjunto de pares variable/valor. Estas colecciones de valores se almacenan en un objeto de la clase Bundle. Para rellenarse el Bundle con valores, puede utilizarse alguna de estas posibilidades:

```
putExtra(String name, boolean value);
putExtra(String name, int value);
putExtra(String name, double value);
putExtra(String name, String value);
```

Para recoger la información en el destino se utiliza:

```
Bundle b = intent.getExtras();
```

2.4.6 Flags

Son metadatos que indican al sistema Android cómo iniciar una actividad y determinar, en parte, el comportamiento del componente que atienda la petición.

2.5. Comunicación entre actividades 

Cuando una actividad ha de lanzar a otra actividad en muchos casos necesita enviarle cierta información. Las actividades son independientes unas de otras, están creadas en clases independientes, así que no comparten variables, por lo que tienen que tener un mecanismo de comunicación entre ellas. Esta comunicación se va a realizar a través de las intenciones.

Cuando queremos realizar una acción, por ejemplo, enviar un mensaje, podemos crear una intención para que haga una de estas acciones:

- Invoque a una actividad que haga el trabajo.
- El sistema invoque la actividad más adecuada.
- La decisión la tome el usuario.

El mecanismo utilizado para realizar el intercambio de datos es el siguiente:

- Para lanzar una actividad B desde la actividad A, usa el siguiente código en A:

```
Intent intent = new Intent(this, MiClase.class);
intent.putExtra("usuario", "Pepito Perez");
intent.putExtra("edad", 27);
startActivity(intent);
```

- En la actividad B, recogemos los datos de la siguiente forma:

```
Bundle extras = getIntent().getExtras();
String s = extras.getString("usuario");
int i = extras.getInt("edad");
```

- Cuando la actividad lanzada (B) termina, si lo desea, podrá enviar datos de vuelta. Para ello añade en la actividad B el siguiente código:

```
Intent intent = new Intent();
intent.putExtra("resultado","valor");
setResult(RESULT_OK, intent);
finish();
```

- En el caso de que el trabajo realizado en la actividad B fuera cancelado, añade este código:

```
Intent intent = new Intent();
setResult(RESULT_CANCEL, intent);
finish();
```

- En la actividad que hizo la llamada (A) has de poder recoger estos datos. Para ello tendremos que lanzar la actividad usando un objeto de la clase *ActivityResultLauncher*, en lugar de *startActivity(Intent)*. Este objeto permite definir un escuchador de evento, que será llamado cuando la actividad lanzada retorne. Para ello introduce el siguiente código:

```
ActivityResultLauncher resultLauncher = registerForActivityResult(
        new ActivityResultContracts.StartActivityForResult(),
        new ActivityResultCallback<ActivityResult>() {
            @Override
            public void onActivityResult(ActivityResult result) {
                if (result.getResultCode() == Activity.RESULT_OK) {
                    Intent intent = result.getData();
                    if (intent!=null){
                        // Código
                    }
                }
            }
        });
```

- Para realizar la llamada reemplaza *startActivity()* por el siguiente código:

```
Intent intent = new Intent(this, MI_CLASE.class);

startActivity(intent);
resultLauncher.launch(intent);
```

Desde la actividad A se podrían llamar a varias actividades. Sin embargo, solo podemos tener un método *onActivityResult()*. Por esta razón, resulta necesario identificar cada actividad lanzada con un código. Así, podremos diferenciar entre los distintos datos devueltos.

3. ANEXO I: TABLA CON INTENCIONES QUE PODEMOS UTILIZAR DE APLICACIONES GOOGLE

Aplicación	URI	Acción	Resultado
Navegador	http://*dirección_web* https://*dirección_web*	VIEW	Abre una ventana de navegador con una URL.
	"" (cadena vacía) http://*dirección_web* https://*dirección_web*	WEB_SEARCH	Realiza una búsqueda web. Se indica la cadena de búsqueda en el extra `SearchManager.QUERY`
Teléfono	tel:*número_teléfono*	CALL	Realiza una llamada de teléfono. Los números válidos se definen en IETF RFC 3966 . Entre estos se incluyen: tel:2125551212 y tel:(212)5551212. Necesitamos el permiso `android.permission.CALL_PHONE`
	tel:*número_teléfono* *voicemail:*	DIAL	Introduce un número sin llegar a realizar la llamada.
Google Maps	geo:*latitud,longitud* geo:*lat,long*?z=*zoom* geo:0,0?q=*dirección* geo:0,0?q=*búsqueda*	VIEW	Abre la aplicación Google Maps para una localización determinada. El campo *z* especifica el nivel de zum.
Google Streetview	google.streetview: cbll=*latitud,longitud*& cbp=1,*yaw,pitch,zoom*& mz=*mapZoom*	VIEW	Abre la aplicación Street View para la ubicación dada. El esquema de URI se basa en la sintaxis que utiliza Google Maps. Solo el campo *cbll* es obligatorio.