ACTIVIDADES E INTENCIONES

Programación multimedia
y dispositivos móviles
2º DAM

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN

- Una actividad en representa una unidad de interacción con el usuario, una pantalla de la aplicación.
- Una aplicación está formada por un conjunto de actividades.
- Toda actividad ha de ser una subclase de Activity.
- Para el cambio de una actividad a otra, utilizaremos la clase Intent.

2. ACTIVIDADES

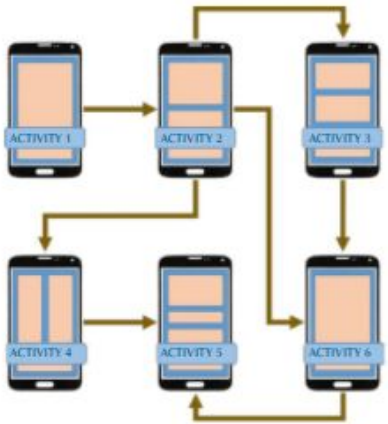
Un Activity es el principal componente de una aplicación en Android, y se encarga de gestionar gran parte de las interacciones con el usuario.

A nivel de lenguaje de programación, una actividad hereda de la clase Activity (o una de sus subclases) y se traduce como una pantalla.

Por tanto, una aplicación tendrá tantas Activities como pantallas distintas tenga. Estas Activities estarán vinculadas entre sí.

Una misma actividad, puede tener distintos componentes, y la interacción con cada uno de ellos, puede llevarnos a una Activity distinta.

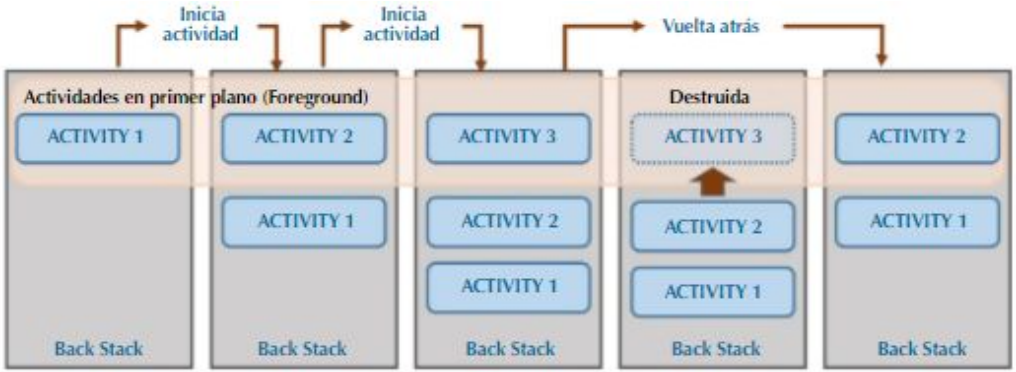
Las Activities funcionan mediante una pila de tipo LIFO (Last In, First Out).



2. ACTIVIDADES

Cuando una actividad inicia otra, la nueva actividad obtiene el foco (se visualiza) y por tanto se coloca en la parte superior de la pila. La actividad anterior permanece en la pila, pero detenida (se está visualizando la segunda actividad).

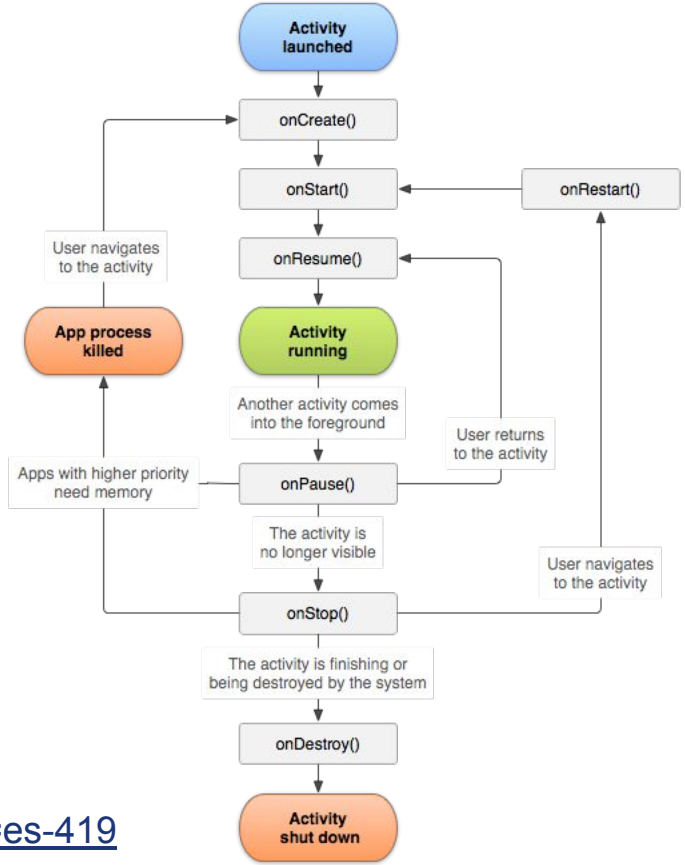
Si un usuario regresa (botón atrás del dispositivo móvil), la actividad que tenía el foco se elimina de la pila, y por tanto, la actividad que se quedó en la pila se restaura y vuelve a coger el foco (se ve la actividad anterior).



3. CICLO DE VIDA DE UNA ACTIVIDAD

Una actividad a lo largo de la ejecución de la aplicación pasa por varios estados:

- Inexistente: aún no se ha indicado que esa actividad deba ser cargada.
- Detenida: no es visible. Hay un cambio de actividad, y la actividad que lanza el intent se queda detenida.
- Pausada: visible pero no accesible. Cuando se abre un diálogo en una actividad.
- Activa: actividad en primer plano, con la que el usuario puede interactuar.



https://developer.android.com/guide/components/activities?hl=es-419

3. CICLO DE VIDA DE UNA ACTIVIDAD

Cuando una actividad se detiene, la actividad queda en la pila y después podrá volver a mostrarse (tal como se ha dejado).

Puede suceder, que si el sistema necesita recursos, elimine la actividad de la memoria, aunque la almacene en la pila. De este modo, al volver atrás podemos encontrarnos con que la interfaz no se muestra tal y como se ha dejado.

Para que esto no suceda, es necesario guardar el estado de la interface gráfica con el método **onSaveInstanceState()**, y después se restaurarán según las instrucciones del método **onRestoreInstanceState()**.

3. CICLO DE VIDA DE UNA ACTIVIDAD

Hay que tener en cuenta, que todos los métodos que intervienen en una actividad se llaman automáticamente, es decir, no hay que invocarlos. Simplemente hay que definir las instrucciones en cada uno de estos métodos.

- **onCreate()**
- **onStart()**
- **onResume()**
- **onPause()**
- **onStop()**
- **onDestroy()**
- **onSaveInstanceState()**
- **onRestoreInstanceState()**

3.1. onCreate

Es llamado cuando la actividad es invocada POR PRIMERA VEZ. Es el momento en que se crean las vistas, se hace reserva de memoria para los datos, se obtienen las referencias e instancias de componentes mediante findViewById(), se suelen hacer las consultas iniciales en las fuentes de datos y se guardan las variables del entorno.

```
setContentView(R.layout.activity_main);

edit = findViewById(R.id.edit);
button = findViewById(R.id.button);

sql = new SQLHelper();

button.setOnClickListener(this);
```

3.2. onStart

Este método se invoca automáticamente al acabar la ejecución de onCreate(). La actividad se hace visible para el usuario, pero aún no se puede interactuar con ella.

La llamada onStart() hace que el usuario pueda ver la actividad mientras la app se prepara para que esta entre en primer plano y se convierta en interactiva.

Suele ser el lugar donde ubicar instrucciones asociadas a la interface de usuario.

- Es un buen lugar para poner un Broadcast Receiver o inicializar algún estado sobre la interfaz de usuario que debe mostrar constantemente siempre que el usuario vuelve a esta actividad.

3.3. onResume

Un método que es llamado cuando la actividad se hace visible para el usuario, pero aún no se puede interactuar con el usuario. Se ejecuta automáticamente después del método onStart().

- En este método se suelen activar sensores, cámara, ejecutar animaciones, actualizar información...

Hay que tener en cuenta que este es el método que se ejecuta después de onRestoreInstaceState(), por lo que si las instrucciones necesitan de los datos guardados, deben ir en este método.

3.4. onPause

Ocurre cuando nuestra actividad pasa a un segundo plano, bien porque está en proceso de destrucción o porque otra actividad pasa a ocupar el primer plano.

- Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.

En este método la actividad es parcialmente visible. Puede ocurrir que vuelva a primer plano (se llame al método onResume()), o que sea parada (se llame al metodo onStop()), haciendolo invisible para el usuario.

3.5. 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, o onDestroy(), si queremos eliminarla definitivamente.

3.6. onRestart

Este método es invocado cuando una actividad parada (anteriormente se ha llamado a onStop()) vuelve a primer plano. Siempre es seguido por el método onStart().

3.7. onDestroy

Es llamado cuando la actividad es definitivamente destruida y eliminada de memoria. En su ejecución se suelen limpiar y liberar recursos, por ejemplo, la cámara.

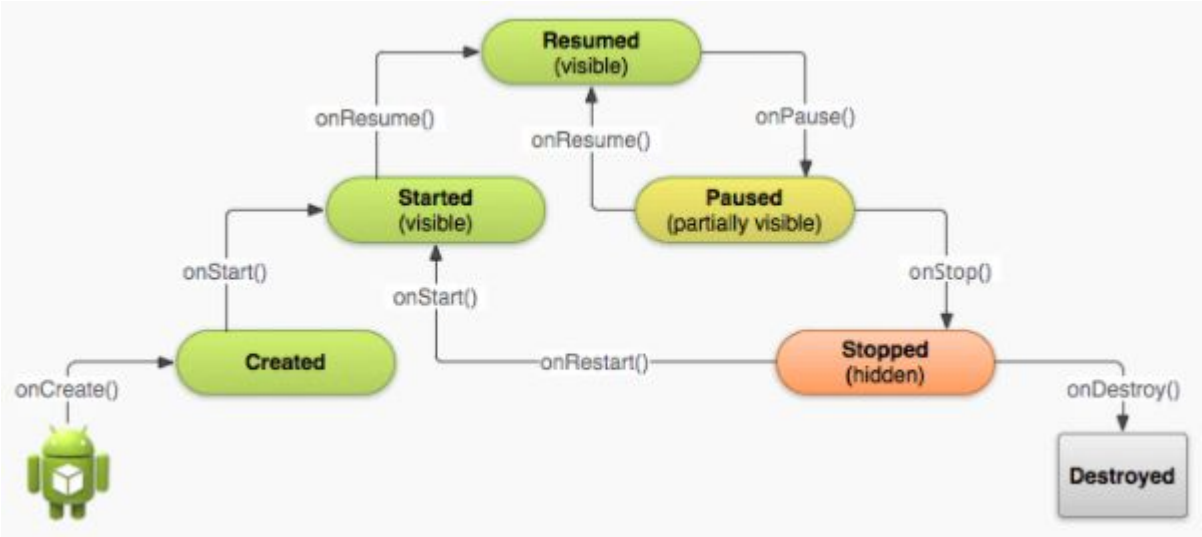
Se puede invocar este método con la instrucción finish().

En el siguiente código de ejemplo, al cambiar de actividad, en vez de cambiar de actividad y dejar la actividad que invoca el cambio en la pila, se elimina.

Si tenemos 2 actividades, y la actividad principal invoca finish() al cambiar a la segunda actividad, cuando el usuario pulse el botón de atrás se cerrará la aplicación.

```
Intent intent = new Intent( packageContext: this, InfoActivity.class);
startActivity(intent);
finish();
```

3.8. RESUMEN CICLO DE VIDA



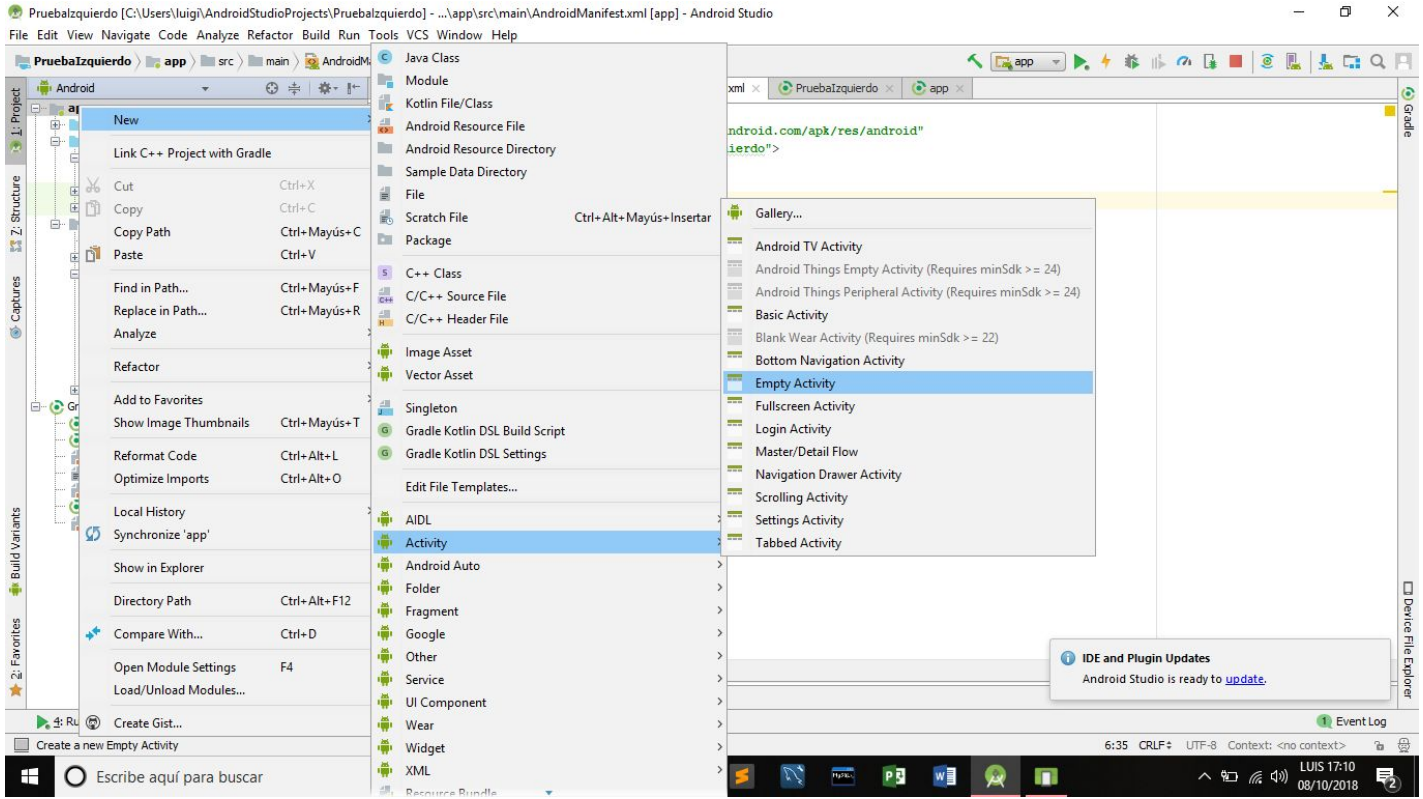
3.8. RESUMEN CICLO DE VIDA

Ciclo de vida completo de una actividad: entre la llamada a onCreate() y la llamada a onDestroy().

Ciclo de vida visible de una actividad: aquel en el que el usuario puede ver la actividad en pantalla. Transcurre desde la llamada a onStart() y la llamada a onStop().

Ciclo de vida en primer plano de una actividad: transcurre entre la llamada a onResume() y la llamada a onPause().

4. CREAR UNA NUEVA ACTIVIDAD



4. CREAR UNA NUEVA ACTIVIDAD

- Cuando creamos una nueva actividad en Android Studio por defecto nos genera dos ficheros:
  - La clase Java de la actividad
  - Su layout escrito en XML
  - Además, nos añade la información correspondiente de la actividad en el AndroidManifest

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/imagen"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".Main2Activity"></activity>
</application>
```

Si no se registra la actividad, cuando vayamos a acceder a ella se parará la aplicación.

5. CONSERVAR EL ESTADO DE UNA ACTIVIDAD

Para guardar el estado, trabajaremos con una nueva clase llamada Bundle.

Esta clase nos permite trabajar con un objeto que puede guardar todas las variables que queramos mediante clave - valor. De esta forma, podemos encapsular todos los datos en un único objeto.

Además de utilizarse para el estado de una actividad, también nos permitirá guardar los valores que necesitemos mover entre actividades.

5. CONSERVAR EL ESTADO DE UNA ACTIVIDAD

Para guardar el estado, trabajaremos con una nueva clase llamada Bundle.

Esta clase nos permite trabajar con un objeto que puede guardar todas las variables que queramos mediante clave - valor. De esta forma, podemos encapsular todos los datos en un único objeto.

Para añadir un valor al Bundle, se realizará mediante el método put junto con el tipo de dato.

Ejemplo: putString, putInt…

Para coger un valor del Bundle, se realizará mediante el método get junto con el tipo de dato.

Ejemplo: getString, getInt...

5. CONSERVAR EL ESTADO DE UNA ACTIVIDAD

```java
@Override
protected void onSaveInstanceState(@NonNull Bundle outState) {
    super.onSaveInstanceState(outState);

    Log.i( tag: "CICLO",  msg: "guardar estado");

    outState.putString("nombre", edit.getText().toString());
    outState.putLong("contador", tiempo);
}

@Override
protected void onRestoreInstanceState(@NonNull Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);

    Log.i( tag: "CICLO",  msg: "recuperar estado");

    edit.setText(savedInstanceState.getString( key: "nombre"));
    tiempo = savedInstanceState.getLong( key: "contador");
}
```

6. COMUNICACIÓN ENTRE ACTIVIDADES

- Las actividades en Android están representadas por clases Java independientes.
- Distintas actividades no comparten información ni variables de forma predeterminada
- Existe un mecanismo de comunicación entre variables basado en *Intenciones* (Intents).
- Los Intents
  - Representan la voluntad de realizar una acción.
  - Permiten lanzar una actividad o un servicio
  - Permiten:
    - Lanzar una actividad dentro de nuestra propia aplicación
    - Lanzar una aplicación del sistema para realizar una determinada acción.
      - Puede ser el sistema quien escoja la aplicación más adecuada
      - O puede ser elegida por el usuario.

6. COMUNICACIÓN ENTRE ACTIVIDADES

Un intent es un sistema de comunicación que permite interactuar entre componentes de la misma o de distintas aplicaciones de Android. Es un elemento básico de comunicación, que se puede utilizar para comenzar otra aplicación, iniciar un servicio, entregar un mensaje...

Existen dos tipos diferentes de intents, según las características y la construcción del mismo:

- Intents explícitos
- Intents implícitos

6.1. INTENTS EXPLÍCITAS

Especifican qué componente se debe iniciar mediante su nombre (el nombre de clase completamente calificado).

Usualmente, el usuario usa una intent explícita para iniciar un componente en su propia aplicación porque conoce el nombre de clase de la actividad o el servicio que desea iniciar.

Por ejemplo, puede utilizarla para iniciar una actividad nueva en respuesta a una acción del usuario o iniciar un servicio para descargar un archivo en segundo plano.

6.1. INTENTS EXPLÍCITAS

Cuando crea una intent explícita para iniciar una actividad o un servicio, la aplicación inicia inmediatamente el componente de la aplicación especificado en el objeto Intent.

Es posible incluir información adicional en una Intent Explícita mediante el método putExtra, de forma que debemos indicar el nombre del objeto y su valor.

- Esta información se almacena en un objeto de la clase *Bundle* que es posible recuperar desde la Activity de destino.

Activity que envía información

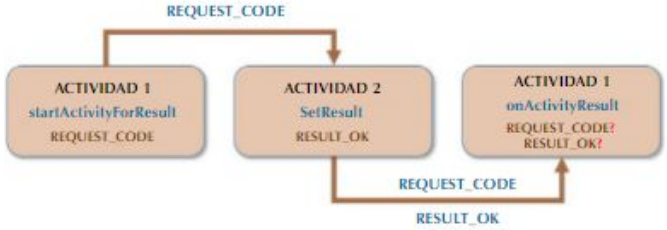
```
Intent intent = new Intent( packageContext: this, MainActivity2.class);
intent.putExtra( name: "nombre", editNombre.getText().toString());
intent.putExtra( name: "edad", editEdad.getText().toString());
startActivity(intent);
```

Activity que recibe información

```
if (getIntent().getExtras() != null) {
    nombre.setText(getIntent().getStringExtra( name: "nombre"));
    edad.setText(String.valueOf(getIntent().getIntExtra( name: "edad",  defaultValue: 0)));
}
```

6.1. INTENTS EXPLÍCITAS

En ocasiones, querremos que una actividad nos devuelva un resultado a la activity desde la que la habíamos llamado. Esto de denominado Propagación.



Activity que envía información

```
Intent intent = new Intent (v.getContext(), LinkActivity.class);
startActivityForResult(intent,0234);
```

```
@Override protected void onActivityResult (int requestCode,
                                           int resultCode, Intent data){
    if (requestCode==0234 && resultCode==RESULT_OK) {
        String res = data.getExtras().getString("resultado");
    }
}
```

Activity que recibe información

```
tr.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent();
        intent.putExtra("resultado","ErrorDanger");
        setResult(RESULT_OK, intent);
        finish();
    }
});
```

6.2. INTENTS IMPLÍCITAS

No se nombra el componente específicamente; pero, en cambio, se declara una acción general para realizar, lo que permite que un componente de otra aplicación la maneje.

Por ejemplo, si desea mostrar al usuario una ubicación en un mapa, puede usar una intent implícita para solicitar que otra aplicación capaz muestre una ubicación específica en un mapa.

Una intent implícita especifica una acción que puede invocar cualquier aplicación en el dispositivo que pueda realizar la acción. El uso de una intent implícita es útil cuando la aplicación no puede realizar la acción pero otras aplicaciones probablemente sí, y el usuario, si tiene más de una aplicación que puede realizar la acción, elegirá una de ellas.

6.2. INTENTS IMPLÍCITAS

Tipo	Descripción
ACTION_SET_ALARM	Crea una alarma.
ACTION_INSERT	Agrega un evento del calendario.
ACTION_IMAGE_CAPTURE	Abre la cámara y captura una imagen.
ACTION_DIAL	Abre la aplicación de teléfono o marcador.
ACTION_CALL	Inicia una llamada telefónica.

https://developer.android.com/guide/components/intents-common.html

6.2. INTENTS IMPLÍCITAS

Ejemplo de una Intent Implícita haciendo una llamada telefónica .

Creamos el intent

```
Intent intent= new Intent(Intent.ACTION_CALL, Uri.parse("tel:" + 666666666));
try {
    startActivity(intent);
} catch (SecurityException e) {
    Toast.makeText( context: this,  text: "Debes aceptar los permisos", Toast.LENGTH_SHORT).show();
}
```

Añadimos el permiso de seguridad en el manifest

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

6.2. INTENTS IMPLÍCITAS

Ejemplo de una Intent Implícita haciendo una llamada telefónica .

Creamos el intent

```
Intent intent= new Intent(Intent.ACTION_CALL, Uri.parse("tel:" + 666666666));
try {
    startActivity(intent);
} catch (SecurityException e) {
    Toast.makeText( context: this,  text: "Debes aceptar los permisos", Toast.LENGTH_SHORT).show();
}
```

Añadimos el permiso de seguridad en el manifest

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

7. SELECTOR DE APPS

Al crear una intent implícita, es posible que el usuario tenga varias aplicaciones y pueda elegir cuál utilizar. Para ello, el Intent debe ser creado con la opción chooser, y comprobar que al menos hay una aplicación en el dispositivo para esta opción.

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_TEXT, editText.getText().toString());
intent.setType("text/plain");

Intent chooser = Intent.createChooser(intent, getResources().getString(R.string.chooser_title));

if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```

8. GESTIÓN DE PERMISOS ANDROID

Para realizar algunas llamadas a servicios del sistema tendremos que declarar los permisos que vamos a necesitar en el Manifest de nuestro proyecto.

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

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="EjemploIntentsImplicitos"
    android:roundIcon="@mipmap/ic_launcher_round"
```

Existen multitud de permisos dependiendo de la acción que vayamos a realizar, podemos encontrar una relación de ellos en :

https://developer.android.com/guide/topics/security/permissions.html?hl=es-419#perm-groups

8. GESTIÓN DE PERMISOS ANDROID

Desde la versión 6.0 de Android los permisos de sistema se dividen en dos grupos

- **Permisos normales** → no ponen en riesgo la privacidad del usuario de forma directa. Si tu app tiene un permiso normal en su manifiesto, el sistema concede el permiso automáticamente. (Tan solo es necesario declararlos en el Manifest)
- **Permisos riesgosos** → pueden permitir que la app acceda a información confidencial del usuario. Si tu app tiene un permiso normal en su manifiesto, el sistema concede el permiso automáticamente. Si tienes un permiso peligroso, el usuario debe autorizar explícitamente a tu app.

8. GESTIÓN DE PERMISOS ANDROID

Solicitar permisos riesgosos en tiempo de ejecución:

- **checkSelfPermission(Context context, String permission)** → nos devuelve PERMISSION_GRANTED si está aceptado, y PERMISSION DENIED si está denegado. Por defecto, está denegado.
- **requestPermissions(Context context, String[] permissions, int requestCode)** → método que muestra una notificación estándar para pedir los permisos del segundo parámetro (permission). Se utiliza un código (requestCode) para identificar el resultado (lo que el usuario permite) en ese diálogo, por sí una misma actividad muestra varias notificaciones de permisos.
- **shouldShowRequestPermissionRationale(Context context, String permission)** → método que devuelve true o false según se pueda o no pedir el permiso al usuario.

https://developer.android.com/training/permissions/requesting.html?hl=es-419

8. GESTIÓN DE PERMISOS ANDROID

- **onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults)** → método que se ejecuta automáticamente al recibir la respuesta del usuario de las notificaciones mostradas en el método **requestPermissions()**.
  - requestCode: el código indicado en requestPermissions().
  - permissions: los permisos indicados en requestPermissions().
  - grantResults: array con los resultados de la interacción para cada permiso indicado por el usuario.

https://developer.android.com/training/permissions/requesting.html?hl=es-419

9. DUDAS Y PREGUNTAS