ACTIVIDADES

Documentación en http://developer.android.com/reference/android/app/Activity.html

Documentación en https://developer.android.com/guide/components/activities/intro-activities

1. COMPONENTES DE UNA APLICACIÓN ANDROID: ACTIVITYS

- Los **componentes** de una aplicación son bloques de código mediante los cuales el sistema puede relacionarse con ella. Cada componente tiene una funcionalidad específica.

- Existen diferentes tipos de componentes entre los que destacan las ***Activity*** (actividades). Otros componentes son:

  - ***Service*** (servicios): componentes sin interfaz gráfica que se ejecutan en segundo plano.

  - ***Broadcast Receiver*** (receptores de emisiones): componentes cuya finalidad es detectar y reaccionar ante determinados mensajes o eventos globales generados por el sistema ("*batería baja*", "*recibido SMS*", …) o por otras aplicaciones, y no dirigidos a ninguna app concreta sino a cualquiera que "quiera" escucharlos.

  - ***Content Provider*** (proveedores de contenido): mecanismo definido en Android para compartir datos entre aplicaciones.

- Las actividades (A*ctivity*) representan el **componente principal** de una aplicación Android. Se puede pensar en una actividad como el elemento análogo a una ventana o pantalla en cualquier otro lenguaje visual. Contiene los elementos o vistas (botones, imágenes, texto…) con los que interacciona el usuario.

- Una Activity, por lo general, ocupa toda la pantalla pero puede ser más pequeña y flotar sobre otras ventanas.

- Cada Activity se corresponde con una **clase Java**, lo que nos permite escribir código para responder a los eventos que ocurren durante todo el ciclo de vida de la Activity.

- La mayoría de las apps contienen varias pantallas. En consecuencia, la mayoría de las apps tienen múltiples actividades.

- Por lo general, una actividad en una app se especifica como la ***actividad principal***, que es la primera pantalla que aparece cuando el usuario inicia la app. Luego, cada actividad puede iniciar otra/s actividad/es a fin de realizar diferentes acciones.

- Una actividad puede iniciar otra de la misma app o bien de otra app diferente.

- Todas las actividades que componen una aplicación deben estar registradas en el archivo **AndroidManifest** (en realidad, todos los componentes deben estar registrados en dicho archivo).

2. ARCHIVO AndroidManifest

Documentación en https://developer.android.com/guide/topics/manifest/manifest-intro

- Es un archivo de configuración que guarda información esencial acerca de la aplicación.

- Su principal tarea es informar al sistema acerca de los componentes de la aplicación. Si un componente no está declarado en el archivo, es como si no existiese. Otras funciones de este archivo son registrar los permisos asociados a la app, declarar las necesidades de software/hardware, etc.

- Ejemplos:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.user.actividades_1" >

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

</manifest>
```

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

    <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.Activities"
        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>
    </application>

</manifest>
```

(Hay más etiquetas que pueden aparecer en el archivo AndroidManifest. Las iremos viendo según sean necesarias)

- Lo fundamental del archivo AndroidManifest es el elemento **<application>**, que permite configurar aspectos básicos de la aplicación como el icono, el tema, el nombre de la aplicación, si la aplicación se debe guardar o no cuando se realice una copia de seguridad del sistema, etc.

- El elemento <**application**> debe contener una entrada **<activity>** para cada una de las actividades que componen una aplicación. Cuando se crea un proyecto, el sistema crea por defecto la Activity principal (***MainActivity***). Y por eso, el archivo AndroidManifest sólo contiene inicialmente un elemento <activity>.

- El elemento <**activity**> debe contener, como mínimo, el atributo "***android:name***". Su valor es el nombre de la clase que representa a la Activity. El carácter punto (.) hace referencia al nombre del paquete.

- El atributo "***android:label***" permite elegir un nombre para mostrar en la barra de títulos, que puede ser el mismo para toda la aplicación o cambiar en cada Activity.

- También se pueden indicar **filtros de intención** (**<intent-filter>**) para que la actividad pueda ser llamada desde otra aplicación.

- Los elementos **<action>** y **<category>**, incluidos dentro de <intent-filter>, se utilizan para indicar la Activity que debería iniciarse cuando el usuario pulse sobre el icono de la aplicación. Son *obligatorios* aunque la aplicación tenga sólo una Activity. En concreto:

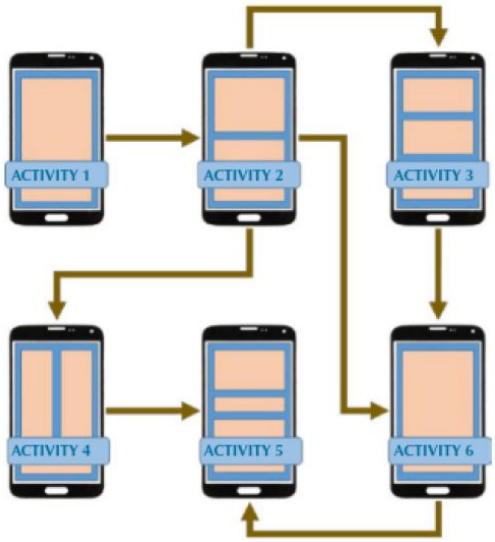
    - *"**android.intent.action.MAIN**"* sirve para que el sistema operativo sepa qué actividad lanzar en primer lugar.

    - *"**android.intent.category.LAUNCHER**"* hace que la aplicación se muestre en la ventana de aplicaciones del dispositivo (*Launcher Screen*).

3. PILA DE ACTIVIDADES

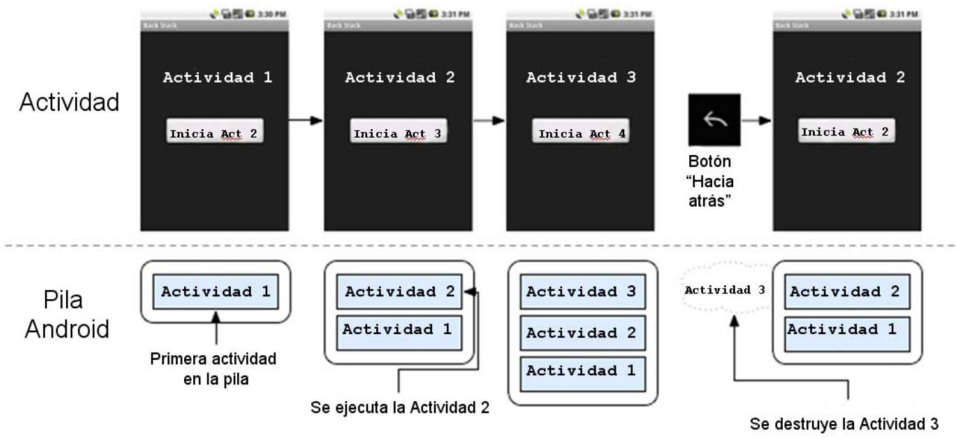
- Una aplicación generalmente consta de múltiples actividades vinculadas entre sí. Las Activity (y sus pantallas asociadas) tendrán una secuencia de aparición según el flujo del programa y las decisiones que vaya tomando el usuario.

- Desde una pantalla principal se podrán realizar desplazamientos a una u otra Activity y, según se va "navegando" por las diferentes pantallas de la app, el dispositivo va "recordando" la secuencia de pantallas visitadas de tal forma que, si se retrocede (utilizando los botones del dispositivo o la lógica de la aplicación), irán apareciendo las pantallas en sentido inverso o cómo se habían visualizado.



(imagen obtenida de **CABRERA RODRIGUEZ: "Programación multimedia y dispositivos móviles". Ed. Síntesis**)

- Android almacena la posición de cada una de las actividades en una pila (Back Stack) de ejecución LIFO ("last in, first out" o "último en entrar, primero en salir").
- Esquema de funcionamiento de la pila de actividades:
  - Cuando se inicia una actividad, ésta pasa al primer plano (Visible). La actividad anterior se detiene y queda en la pila "tapada" por la nueva actividad, la cual pasa a obtener el foco.
  - Al pulsar la tecla de retroceso del móvil (***back***), se destruye la actividad actual y la actividad previa, que se encontraba debajo de ella en la pila, se restaura y vuelve a coger el foco.
  - Si se continúa presionando el botón *back*, se irán eliminando sucesivamente activities hasta llegar a la pantalla de inicio.
  - Cuando todas las activities se han eliminado de la pila, la tarea deja de existir y desaparece de la memoria.
  - El siguiente esquema representa cómo cambia la pila de Android al ir abriendo actividades y al pulsar el botón "***back***":



- El conjunto de actividades que tienen una relación lógica y que se encuentran en la pila se denomina ***tarea*** (task).

- Desde que iniciamos una actividad hasta que salimos de ella, dicha actividad pasa por diferentes etapas o estados. Para Android el estado de la actividad puede ser:
  - *Inexistente*: no ocupa memoria por lo que, obviamente, no se puede interaccionar con ella.
  - *Detenida*: sí ocupa memoria, pero no es visible y, por tanto, tampoco se puede acceder a ella.
  - *Pausada*: es visible, pero aún no se puede interactuar con ella.
  - *Activa*: está en primer plano y el usuario puede interaccionar con ella.
- Estos estados suelen ocurrir en sucesión creciente (cuando la actividad se pone en marcha) o bien decreciente (cuando finaliza y se destruye). Por eso, el conjunto de estados de una actividad se conoce como "***ciclo de vida***".

4. CICLO DE VIDA DE UNA ACTIVIDAD

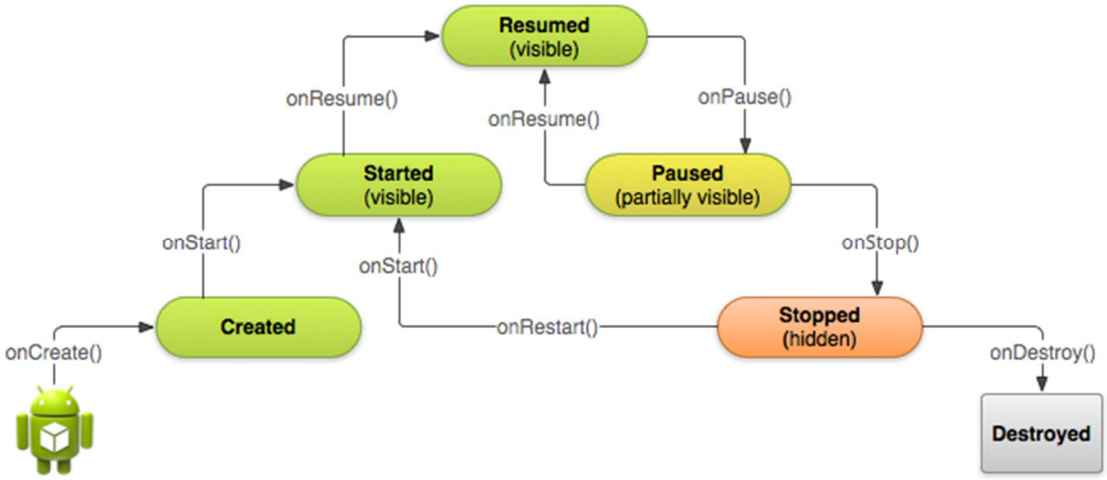
- El ciclo de vida de una actividad ocurre entre las llamadas a los métodos ***onCreate()*** y ***onDestroy()***.
  - En el método ***onCreate()*** de la actividad se realiza la reserva de memoria, el diseño de la interfaz de usuario y se recupera el estado de la sesión anterior.
  - En el método ***onDestroy()*** se liberan todos los recursos usados con anterioridad por dicha aplicación.
- Durante su ciclo de vida, una actividad pasa por diferentes estados:

- **Resumed**: la actividad está en el primer plano de la pantalla (**visible**): **ejecutándose**.

- **Paused**: la actividad es **parcialmente visible**, pero hay otra actividad en primer plano, que no ocupa toda la pantalla, por encima de la primera. La actividad pausada se mantiene en memoria, aunque el sistema operativo puede eliminarla en caso de que necesite dicha memoria.

- **Stopped**: la actividad está completamente **oculta** por una nueva actividad. La actividad detenida también se mantiene en memoria. Sin embargo, el usuario ya no la ve y el sistema operativo también puede eliminarla cuando necesite memoria para otra tarea.

- **Created** y **started**: Desde el estado en que se lanza una actividad hasta que llega al estado **Resumed (Running)**, ésta pasa por los estados **Created** y **Started**.



- Cada uno de los cambios de estado de una actividad tiene asociado un método (método ***callback)*** que será llamado en el momento de producirse dicho cambio.

- Ejemplo:

  - Desde que una actividad se lanza y hasta que llega al estado "***resumed***" se ejecutan los métodos callback ***onCreate()***, ***onStart()*** y ***onResume()***, por este orden**.**

  - Si una actividad se está ejecutando en primer plano (estado "***resumed***"), y pasa al estado "***paused***", en este cambio se ejecutará el método callback ***onPause()***.

  - Desde el estado "***paused***", una actividad puede volver a situarse en primer plano, en cuyo caso se ejecutará el método callback ***onResume()***. O también puede hacerse invisible para el usuario: estado "***stopped***". En este caso se ejecutará ***onStop()***.

- Una actividad puede pasar del estado "***stopped***" a "***resumed***", porque vuelve a primer plano desde la pila de actividades. En este caso, los métodos ejecutados son ***onRestart()***, de nuevo ***onStart()***, y finalmente ***onResume()***.

- Estos métodos **callback** capturan los cambios de estado que se van produciendo en la actividad y pueden ser sobreescritos para que realicen las operaciones que nosotros queramos.

- Es importante tener claro que **el programador no realiza nunca las llamadas a onCreate() ni a ninguno de los otros métodos del ciclo de vida de una Activity de forma explícita**. El programador sólo los sobreescribe en las subclases de Activity y **es el sistema operativo quien los llama en el momento apropiado**.

- La implementación de estos métodos siempre debe incluir la **llamada al método de la clase superior** (superclase) antes de ejecutar cualquier otra sentencia. Por ejemplo, para el método ***onCreate()***, la primera sentencia es "**super.onCreate();**".

- Una actividad puede llegar al estado "***destroyed*****"** de diferentes formas:

  - Está en primer plano ("***resumed***") y se pulsa el botón "atrás". En este caso llega al método "***destroyed*****"**, pasando por "***paused*****" y** "***stopped***", y se ejecutarán todos los métodos implicados: ***onPause()***, ***onStop()*** y ***onDestroy()***.
  - Se destruye explícitamente mediante la codificación del método ***finish()***.
  - Desde el administrador de aplicaciones.

- **Cuando se modifica la configuración del terminal en tiempo de ejecución, el sistema destruye la actividad actual y la vuelve a crear** para que se adapte lo más posible a las nuevas características ya que puede que haya recursos alternativos mejor adaptados a la nueva configuración.

  El cambio de configuración más habitual en tiempo de ejecución es el cambio de la orientación del dispositivo: si hay una actividad en primer plano, ésta se destruye y se vuelve a lanzar. Como consecuencia, se volverá a ejecutar el método ***onCreate()***. Y se perderán los valores de las vistas salvo que estas tengan creado un ID: **android:id="@+id/**..."**,** así como los valores internos de nuestra aplicación.

5. EJEMPLO

- Al ejecutar la aplicación por primera vez, observamos que se llama a los métodos:

- Si pulsamos la tecla "Home", para que la actividad pase a segundo plano, se ejecutan:

- Si pulsamos el icono de nuestra aplicación desde la ventana de lanzamiento, se ejecutan:

- Si pulsamos la tecla de “Pantalla atrás”, se ejecutan:

- Al girar el dispositivo también se destruye la actividad y se inicia desde cero. Pero si hemos tecleado algo, por ejemplo en una caja de texto, veremos que el sistema guarda automáticamente ese valor y lo muestra al reiniciar.
  Eso es así porque el sistema guarda en un objeto de tipo **Bundle** (conjunto de pares “parámetro”- “valor”) el estado de cada vista de la Activity que tenga creado un identificador "@+id/"
  Ese objeto Bundle es recuperado en el método ***onCreate()*** cuando se inicia de nuevo la actividad.

6. INTENTS

Documentación: http://developer.android.com/reference/android/content/Intent.html

- Para pasar de una actividad a otra se utilizan los ***Intent***.
- Un **Intent** es lo que permite a una aplicación manifestar la "intención" de que desea hacer algo. Por ejemplo:
  - Abrir una nueva Activity (pantalla), de la misma aplicación o de otra distinta.
  - Pasar datos de una Activity a otra, de su misma aplicación o de otra distinta.
  - Interconectar otros componentes de la misma o distinta aplicación.
- Los intents se utilizan para arrancar componentes de dos formas:
  - **Explícita**: invocando la clase Java del componente que queremos ejecutar. Normalmente, esto se usa para invocar componentes de una misma aplicación.
  - **Implícita**: invocando la acción y los datos sobre los que aplicar dicha acción. Android selecciona, en tiempo de ejecución, la actividad receptora que cumple mejor con la acción y los datos solicitados.
- En algunos casos, se puede iniciar una subactividad para recibir un resultado, en cuyo caso esta **subactividad devuelve el resultado en otra nueva Intent.**

- Para arrancar una Activity sin esperar una respuesta de la subactividad iniciada, debemos usar el siguiente método:

  ```
  startActivity(unIntent);
  ```

  Para arrancar una Activity y esperar una respuesta de la subactividad iniciada, debemos usar la siguiente función:

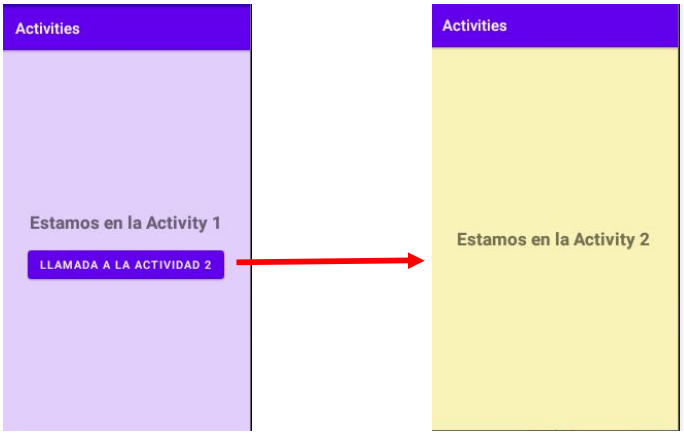
  ```
  startActivityForResult(unIntent, INTENT_COD);
  ```

  Ambos métodos se pueden usar tanto en las invocaciones explícitas como implícitas. La diferencia radica en que el primero inicia la subactividad y no espera respuesta de ésta; el segundo método espera recibir una respuesta de la ejecución de la subactividad.

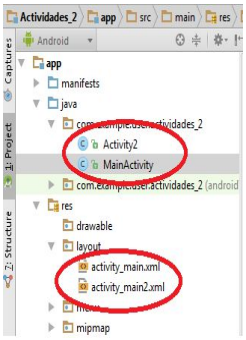
- El fichero "**AndroidManifest**" debe incluir todas las actividades (y demás componentes de una aplicación), ya que, si no lo hacemos, no son visibles para el sistema y, en consecuencia, no se pueden ejecutar. La actividad principal ya debe aparecer puesto que se creó de forma automática al crear el nuevo proyecto Android, pero debemos añadir las demás.

6.1. Lanzar una segunda actividad de la misma aplicación.

Para ello vamos a crear un proyecto llamado Activities.



Estructura del proyecto



Archivo de layout para la pantalla inicial (activity_main.xml)

Contiene una *TextView* y un *Button*.

Archivo de layout para la segunda pantalla (activity_main2.xml)

Contiene una *TextView*.

Código de la Activity principal

```
package com.example.activities;

    (…)

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClickCambiarActividad (View v){
        Intent miIntent = new Intent(this, Activity2.class);
        startActivity(miIntent);
    }
}
```

Comentario:

Para llamar a la segunda actividad, creamos un objeto de tipo **Intent**. La clase **Intent** tiene diferentes constructores:

Public constructors
`Intent()` Create an empty intent.
`Intent(Intent o)` Copy constructor.
`Intent(String action)` Create an intent with a given action.
`Intent(String action, Uri uri)` Create an intent with a given action and for a given data url.
`Intent(Context packageContext, Class<?> cls)` Create an intent for a specific component.
`Intent(String action, Uri uri, Context packageContext, Class<?> cls)` Create an intent for a specific component with a specified action and data.

Al constructor de la clase Intent le pasamos una **referencia a la propia actividad**, y la **clase de la actividad llamada**. Estamos haciendo uso de un intent explícito:

```
Intent miIntent = new Intent(this, Activity2.class);
```

Para hacer referencia a la propia actividad podemos usar "**this**" (si estuviésemos dentro de una clase anónima, "this" haría referencia a ella en lugar de a la actividad principal. Y podríamos emplear el método ***getApplicationContext()*****,** para obtener el contexto o la referencia a la actividad).

La siguiente sentencia lanza la segunda Activity mediante la llamada al método ***startActivity(intent)***, que lleva como parámetro el intent que se ha implementado antes:

```
startActivity(miIntent);
```

Código de la Activity secundaria

```
package com.example.activities;

import android.app.Activity;
import android.os.Bundle;

public class Activity2 extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);
    }
}
```

Comentario:

En este caso, la segunda actividad simplemente muestra el segundo layout.

6.2. Lanzar una segunda actividad de la misma aplicación, con envío de datos.

Para ello vamos a modificar el proyecto anterior, de forma que al llamar a la nueva actividad le enviemos un dato, por ejemplo, la cadena de caracteres "La Activity 1 envía este mensaje a la actividad 3", como muestran las siguientes capturas.

Código de la Activity principal

```
package com.example.activities;

(…)

public class MainActivity extends AppCompatActivity {

    private String datoEnviado = "Activity 1 envía este mensaje a la Activity 3";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClickCambiarActividad (View v){
        Intent miIntent = new Intent(this, Activity3.class);
        miIntent.putExtra("dato", datoEnviado);
        startActivity(miIntent);
    }
}
```

Comentario:

Llamamos al método ***putExtra()*** de la clase Intent. El método ***putExtra()*** permite añadir los datos que queremos enviar. Tiene dos parámetros, que se interpretan como una pareja "clave-valor". En este ejemplo:

Clave → "dato"
Valor → datoEnviado

```
miIntent.putExtra("dato", datoEnviado);
```

Código de la Activity secundaria

```
package com.example.activities;

import ...;


public class MainActivity3 extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main3);

        Intent intent=getIntent();
        String datoRecibido=intent.getStringExtra("mensaje");
        TextView tvRecibida=findViewById(R.id.textViewRecibida);
        tvRecibida.setText(datoRecibido);
    }
}
```

Comentario:

Los datos enviados pueden ser recuperados luego en la nueva Activity. Para ello, en primer lugar accedemos al intent que ha originado la actividad actual, mediante el método ***getIntent()***, de la clase Activity:

Intent	getIntent() Return the intent that started this activity.

Para recuperar el valor del dato recibido hemos utilizado el método ***getStringExtra()***, de la clase Intent:

String	getStringExtra(String name) Retrieve extended data from the intent.

```
datoRecibido = intent.getStringExtra("mensaje");
```

Por último, mostramos una TextView con el string recibido

Concepto y uso de objeto Bundle

Tanto el envío como la recuperación de un dato se pueden hacer también mediante un objeto de la clase **Bundle** (http://developer.android.com/reference/android/os/Bundle.html):

Envío:

```
datoAEnviar="Activity 1 envía este mensaje en un bundle a la Activity 4";
Intent intent=new Intent(this, MainActivity4.class);
// definimos un objeto de tipo Bundle:
Bundle bundle=new Bundle();
// añadimos el String al bundle
bundle.putString("mensaje", datoAEnviar);
// asociamos el objeto bundle al intent
intent.putExtras(bundle);
startActivity(intent);
```

Recuperación:

```
Intent intent=getIntent();
//recuperamos el objeto bundle que se le ha pasado con el intent
Bundle bundle=intent.getExtras();
//obtenemos el dato con el método asociado al tipo de dicho dato
String datoRecibido=bundle.getString("mensaje");
//en una sola sentencia
String datoRecibido2=getIntent().getExtras().getString("mensaje");
TextView tvRecibida=findViewById(R.id.textViewRecibida);
tvRecibida.setText(datoRecibido2);
```

Comentario:

Para recuperar los datos que se habían enviado a través del objeto Bundle, utilizamos el método ***getExtras()*** de la clase Intent. Este método devuelve un objeto Bundle.

`Bundle`	`getExtras()` Retrieves a map of extended data from the intent.

A partir de dicho objeto Bundle, recuperamos el dato mediante el método get<type>, donde el tipo puede ser String, int, etc. (es decir, get<type> sería concretamente un método ***getString()***, ***getInt()***, etc.).

```
Bundle unBundle = getIntent().getExtras();
datoRecibido = unBundle.getString("dato");
```

6.3 Lanzar una segunda actividad de la misma aplicación esperando respuesta.

Documentación: http://developer.android.com/training/basics/intents/result.html

- Para llamar a una actividad con la intención de obtener datos de ella, hay que utilizar el método ***startActivityForResult()***.
- La subactividad devuelve el resultado por medio de otro objeto de tipo **Intent**, de la misma forma que en los ejemplos anteriores. Para devolver el resultado a la actividad que realizó la llamada, hay que utilizar el método ***setResult()***.

`final void`	`setResult(int resultCode, Intent data)` Call this to set the result that your activity will return to its caller.
`final void`	`setResult(int resultCode)` Call this to set the result that your activity will return to its caller.

- La actividad principal recibe los datos retornados por la subactividad mediante el método callback ***onActivityResult()***.
- Para ejemplificar esto, vamos a ampliar el proyecto anterior. Consistirá en que la actividad 1 invoque a la actividad 5 con intención de que ésta le retorne un dato. Y la actividad 5 va a retornar también una cadena de caracteres a la actividad1.

Archivo de layout para la pantalla inicial (activity_main.xml)

Añadimos una TextView donde recuperaremos el dato retornado.

Archivo de layout para la pantalla 5 (activity_main5.xml)

Añadimos un Button para enviar un dato desde la Activity5.

Código de la Activity principal

```
public class MainActivity extends AppCompatActivity {

    private static final int CODIGO_LLAMADA_CON_RESPUESTA = 1;
    private String datoAEnviar;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity main);
    }
(…)

datoAEnviar="Activity 1 envía este mensaje a la Activity 5 esperando respuesta";
Intent intent=new Intent(this, MainActivity5.class);
intent.putExtra("mensaje", datoAEnviar);
startActivityForResult(intent, CODIGO_LLAMADA_CON_RESPUESTA);

(…)

@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(requestCode==CODIGO_LLAMADA_CON_RESPUESTA){
        TextView tvRecibida=findViewById(R.id.textViewRetornada);
        if(resultCode==RESULT_OK){
            String strRecibido=data.getStringExtra("mensaje_devuelto");
            tvRecibida.setText(strRecibido);
        }else{
            //probamos esto como caso en que el user pulsó el btn back --> no llega el OK
            tvRecibida.setText("La actividad 5 se cerró de forma inesperada");
        }
    }
}
```

Comentario:

La actividad secundaria se llamó con el método

startActivityForResult(Intent intent, int requestCode)

Este método indica que esperamos que la subactividad devuelva un resultado cando termine.

El número que se pasa como segundo parámetro es el código de la petición. Se trata de un **valor entero** definido por el desarrollador (debería ser una constante), y es el mismo que nos va a devolver la actividad "hija" cuando finalice su ejecución. Su función es identificar la actividad que está enviando el resultado, porque podíamos haber llamado a varias subactividades.

```
startActivityForResult(miIntent, CODIGO);
```

Cuando finaliza la subactividad y se vuelve a la actividad principal, en ésta, sobreescribiendo el método ***onActivityResult(int requestCode, int resultCode, Intent data)*** podemos comprobar que número nos devuelve la subactividad que nos pasa el control y actuar según deseemos.

Parámetros de onActivityResult():

- ***int requestCode***: valor entero inicialmente suministrado desde el método startActivityForResult(), que permite identificar de dónde proceden los resultados.
- ***int resultCode***: valor entero devuelto por la subactividad a través de su método setResult().
- ***Intent data***: intent que permite retornar los datos.

La actividad principal recibe la llamada a ***onActivityResult()*** inmediatamente antes de los métodos callback ***onRestart()*** y ***onResume()***, cuando la actividad está relanzándose.

Código de la Activity secundaria

```
public void onClickBtnVolver(View view) {
    String datoDeRespuesta="La actividad 5 está respondiendo a la llamada";
    Bundle b=new Bundle();
    //insertamos el string de retorno en el bundle correspondiente
    b.putString("mensaje_devuelto", datoDeRespuesta);
    Intent i=new Intent();
    //asociamos el objeto bundle al intent que utilizaremos para enviar datos de retorno
    i.putExtras(b);
    //realizamos el envío con el código de respuesta
    setResult(RESULT_OK, i);
    //finalizamos la ejecución de esta activity
    finish();
}
```

Comentario:

Para devolver el resultado a la actividad que realiza la llamada se utiliza el método ***setResult()***

`final void`	`setResult(int resultCode, Intent data)` Call this to set the result that your activity will return to its caller.
`final void`	`setResult(int resultCode)` Call this to set the result that your activity will return to its caller.

(En este ejemplo, no es un resultado en sí, sino que se trata de una cadena de caracteres definida en la activity)

El primer parámetro indica un código de resultado. Tiene que incluírse siempre. En general, es una de dos constantes predeterminadas: ***RESULT_OK*** o ***RESULT_CANCELED***.

- ***RESULT_OK*** indica que lo que se tenía que hacer en la actividad secundaria se realizó correctamente.

- Si se quiere indicar que en la actividad secundaria falló algo, utilizaremos la constante ***RESULT_CANCELED.***
  El resultado se establece en ***RESULT_CANCELED*** de forma predeterminada. De este modo, si el usuario presiona el botón *"Atrás"* antes de completar la acción y antes de que se establezca el resultado, la actividad original recibirá el resultado de "cancelada".

Como segundo parámetro, si fuese necesario, se pueden proporcionar datos adicionales con un Intent

Por último, hay que obligar a que la segunda actividad finalice mediante el método ***finish()***.

6.4 Lanzar una actividad de otra aplicación

- El uso de intents también permite que la app inicie una actividad que esté contenida en otra aplicación.

- Se puede hacer mediante el método ***setClaseName()***, de la forma siguiente:

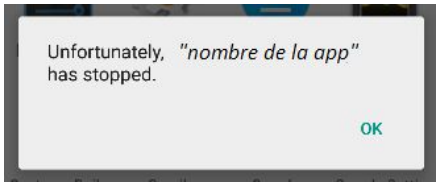
`Intent`	`setClassName(String packageName, String className)` Convenience for calling `setComponent(ComponentName)` with an explicit application package name and class name.

El método ***setClassName (String paquete, String clase)*** contiene como primer parámetro el **paquete** en donde está la clase que corresponde con la activity que queremos lanzar; y, como segundo parámetro, el nombre de la propia **clase**.

- Ejemplo:

```
intent.setClassName("com.example.nombre_del_paquete", "com.example.
nombre_del_paquete.Actividad_llamada")
```

- Si en el dispositivo no existiese la actividad a la que queremos llamar con nuestro intent, nuestra aplicación fallaría (se produce una ***ActivityNotFoundException***) y mostraría un mensaje de error como el siguiente:



- Podemos verificar que haya una actividad disponible que pueda responder al intent haciendo uso del ***PackageManager*** y su método ***queryIntentActivities()***, como muestra el fragmento de código siguiente:

```
PackageManager pm = getPackageManager();
List actividadesPosibles = pm.queryIntentActivities(i, PackageManager.MATCH_DEFAULT_ONLY);
if (actividadesPosibles.size()>0){
   startActivity(i);
}
else{
   Toast.makeText(MainActivity.this, "Ninguna actividad puede realizar esta acción", Toast.LENGTH_SHORT).show();
}
```

  - El ***PackageManager*** "conoce" todos los componentes instalados en el dispositivo, incluidas todas las activity. Creamos una instancia de la clase PackageManager mediante ***getPackageManager()***.
  - Invocando ***queryIntentActivities(intent, int)*** obtenemos una lista de actividades que se corresponden con el intent que hemos configurado. Si el objeto ***List*** no está vacío, es posible usar la intent de forma segura.
  - La constante ***MATCH_DEFAULT_ONLY*** hace que la búsqueda se realice entre las activity con la indicación CATEGORY_DEFAULT (igual que hace, de forma defectiva, el método startActivity(Intent)).

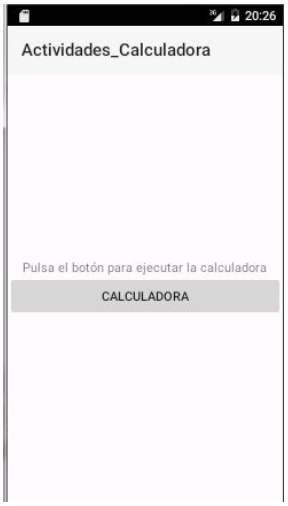
- Conseguimos el mismo resultado manejando la excepción producida:

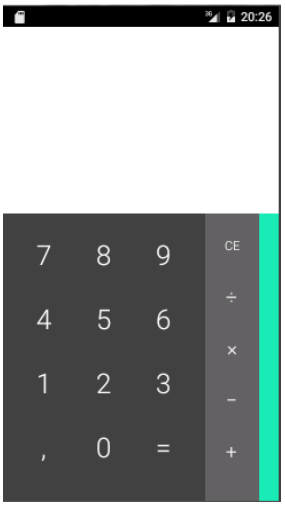
```
try {
   startActivity(intent);
} catch (ActivityNotFoundException e) {
   Toast.makeText(MainActivity.this, "Ninguna actividad puede realizar esta acción", Toast.LENGTH_SHORT).show();
}
```

Documentación: https://developer.android.com/training/basics/intents/sending.html

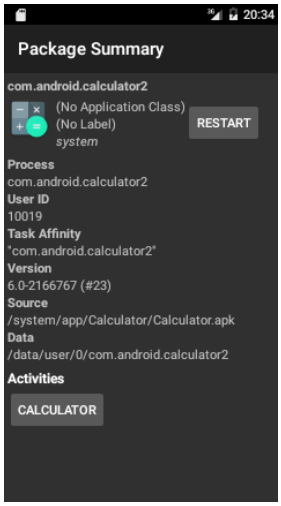
6.5 Lanzar una actividad de una aplicación del sistema

- Una aplicación también puede llamar a diferentes aplicaciones incorporadas en el dispositivo Android.
- Por ejemplo, vamos a solicitar la ejecución de la aplicación de la calculadora como respuesta a la pulsación de un botón, como se ve en las imágenes siguientes:





- La información necesaria (nombre del paquete y de la clase) la podemos obtener desde la aplicación ***Dev Tools/Package Browser/Calculadora***:

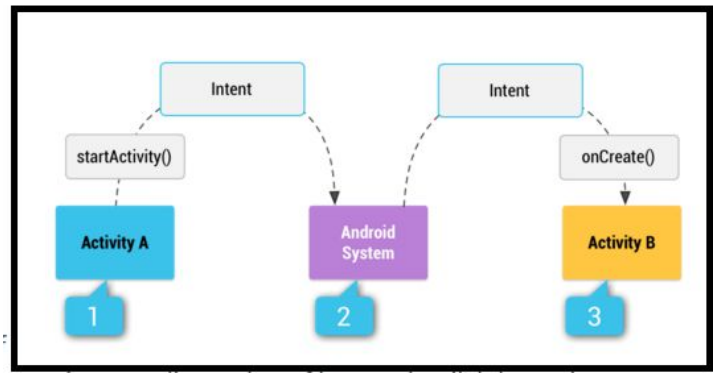


- El código necesario para esto sería:

```
Intent i = new Intent();
i.setClassName("com.android.calculator2", "com.android.calculator2.Calculator");
startActivity(i);
```

6.6 Intents implícitos.

- Un componente Android, como una Activity, puede ser lanzado de forma implícita cuando se indica la **acción** que se desea realizar y, si es el caso, los **datos** sobre los que se va a realizar dicha acción.
- A diferencia de los intents explícitos, en los intents implícitos **no se especifica el nombre de la actividad que va a responder al intent**.
- En los intents implícitos simplemente informamos al sistema de la acción que deseamos realizar y **es el sistema el que busca cuál es el componente** (en nuestro caso, estamos hablando de activities) **más adecuado** para realizar dicha acción.
- En otras palabras, no se especifica una Activity concreta: el sistema escoge la actividad que va a atender a la llamada según la acción que hayamos indicado en el intent
- Por ejemplo:
  - Si queremos ver el contenido de una página web, el sistema propone el navegador con el que mostrarla o, si dispone de más de uno instalado en el dispositivo, le preguntará al usuario con cuál hacerlo.
  - Si deseamos mostrar al usuario una ubicación en un mapa, podemos usar una intent implícita para solicitar que se ejecute una Activity capaz de realizar dicha acción.



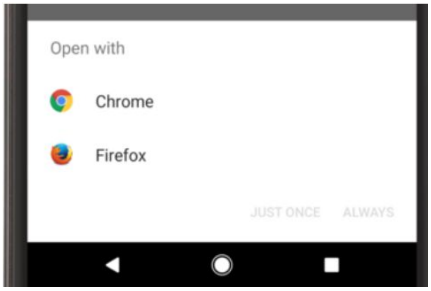
Documentación: http://developer.android.com/intl/es/guide/components/intents-filters.html

- La ilustración representa el modo en que una intención implícita es entregada a través del sistema para iniciar otra actividad:
  - [1] La Actividad A crea un intent con una descripción de la acción deseada y la pasa al método *startActivity()*.
  - [2] El sistema Android busca en todas las aplicaciones un filtro de intención que coincida con la intención solicitada. Cuando se encuentra una coincidencia,

- [3] el sistema inicia la actividad correspondiente (Actividad B) invocando su método *onCreate()* y pasándole la intención.

- El sistema encuentra el componente apropiado para atender a la llamada comparando la información del intent con el contenido de los **filtros de intención** declarados en el archivo de manifiesto de todas las aplicaciones existentes en el dispositivo. Pueden ocurrir varias cosas:

  - Si la intención coincide con lo indicado en un filtro de intención, el sistema inicia ese componente.
  - Si varios filtros de intención son compatibles, el sistema muestra un cuadro de diálogo para que el usuario pueda elegir qué aplicación utiliza:



  - Si el usuario no tiene ninguna de las aplicaciones que se encargan de la intención implícita que se envió al método *startActivity()*, la llamada fallará y la aplicación se bloqueará. Obviamente, es responsabilidad del programador testear la existencia o no de alguna aplicación que responda al intent antes de solicitar la ejecución del método *startActivity()*

- Para crear un Intent implícito se indicará la **acción** deseada y, en la mayor parte de los casos, los **datos** correspondientes a dicha acción. Por ejemplo, si lo que deseamos es llamar a un determinado número de teléfono indicaremos la acción (realizar una llamada) y el número deseado.
- La clase Intent contiene **constantes de acción** para indicar qué es lo que se desea hacer. P. ej.
  - **ACTION_VIEW**, para mostrar datos al usuario.
  - **ACTION_EDIT**, para editar datos.
  - **ACTION_PICK,** para seleccionar un ítem de un conjunto de datos.
  - **ACTION_CALL,** para realizar una llamada telefónica, etc.
- El ejemplo de la llamada telefónica a un determinado número quedaría:
  - Acción: **Intent.ACTION_CALL**
  - Dato: un **objeto Uri**, cuyo contenido sea el número de teléfono

```
i = new Intent(Intent.ACTION_CALL, Uri.parse("tel:(+34)981445566"));
```

(Ver constructores en página 10)

También se pueden emplear los métodos ***setAction()*** y ***setData()***

Intent	setAction(String action) Set the general action to be performed.
Intent	setData(Uri data) Set the data this intent is operating on.

```
i.setAction(Intent.ACTION_CALL);
i.setData(Uri.parse("tel:(+34)981445566"));
```

- En http://developer.android.com/intl/es/reference/android/content/Intent.html se recoge la lista de acciones disponibles. La siguiente captura muestra una parte de dicha lista de acciones.

Standard Activity Actions

These are the current standard actions that Intent defines for launching activities (usually through `startActivity(Intent)`. The most important, and by far most frequently used, are `ACTION_MAIN` and `ACTION_EDIT`.

- `ACTION_MAIN`
- `ACTION_VIEW`
- `ACTION_ATTACH_DATA`
- `ACTION_EDIT`
- `ACTION_PICK`
- `ACTION_CHOOSER`
- `ACTION_GET_CONTENT`
- `ACTION_DIAL`
- `ACTION_CALL`
- `ACTION_SEND`
- `ACTION_SENDTO`
- `ACTION_ANSWER`

(...)

- En developer.android.com/guide/appendix/g-app-intents.html se muestran también varios ejemplos de uso común.

- Para verificar que alguna actividad recibirá el intent, podemos usar el método ***resolveActivity()*** de la clase **Intent**. Si el resultado no es nulo, hay al menos una aplicación que puede administrar el intent y es seguro llamar a ***startActivity()***. En caso contrario, llamar a startActivity() provocará un error de ejecución.

ComponentName	resolveActivity(PackageManager pm) Return the Activity component that should be used to handle this intent.

Código:

```
if (i.resolveActivity(getPackageManager()) != null)
{//resoluble
    startActivity(i);
```

7. PERMISOS

Documentación: https://developer.android.com/guide/topics/permissions/overview?hl=es

- Cuando instalamos una aplicación en un dispositivo real puede ocurrir que se necesite acceder a características que exigen algún tipo de permiso. En ese caso, el proceso de instalación pregunta si estamos dispuestos a dar ese tipo de permiso para que la aplicación pueda funcionar con todas sus características.
- En el fichero **AndroidManifest.xml** se declaran los permisos que necesita la aplicación para poder utilizar funciones protegidas, como contactos, cámara, memoria usb, gps, etc.
- Estos permisos se declaran mediante una o varias etiquetas **<uses-permission>.**
- Ejemplos:

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

- **Modificación en la gestión de permisos a partir de la versión 6.0** (API 23 – MarshMallow):
  - Lo anterior es válido hasta la API 23.
  - Esta forma de conceder permisos conlleva que, una vez concedido un permiso, el acceso al mismo ya no es controlable por el usuario, que tan solo puede desinstalar la aplicación.
  - A partir de Android 6.0 (nivel de API 23), los usuarios conceden permisos a las apps **mientras se ejecutan**, **no cuando instalan la app**. Este enfoque simplifica el proceso de instalación de la app, ya que el usuario no necesita conceder permisos cuando instala o actualiza la app. También brinda al usuario mayor control sobre la funcionalidad de la app; por ejemplo, un usuario podría optar por proporcionar a una app de cámara el acceso a ésta, pero no permitirle el acceso a la ubicación del dispositivo. El usuario puede revocar los permisos en cualquier momento desde la pantalla de configuración de la app.

- Los permisos del sistema se dividen básicamente en dos categorías, "***normal***" y "***peligroso***":
  - Las acciones que no presentan un gran riesgo para la privacidad del usuario o para el funcionamiento del dispositivo necesitan permisos de los considerados "**normales**".

    En este caso, basta con poner el **<uses-permission>** correspondiente en el archivo AndroidManifest de la aplicación.

    Si una app tiene un permiso normal en su manifiesto, el sistema concede el permiso automáticamente.

    Un ejemplo es el acceso a Internet

  - Para aquellas otras acciones con más riesgo, porque podrían afectar a la privacidad del usuario o al funcionamiento normal del dispositivo, el sistema solicita al usuario que otorgue explícitamente esos permisos, considerados como "peligrosos".

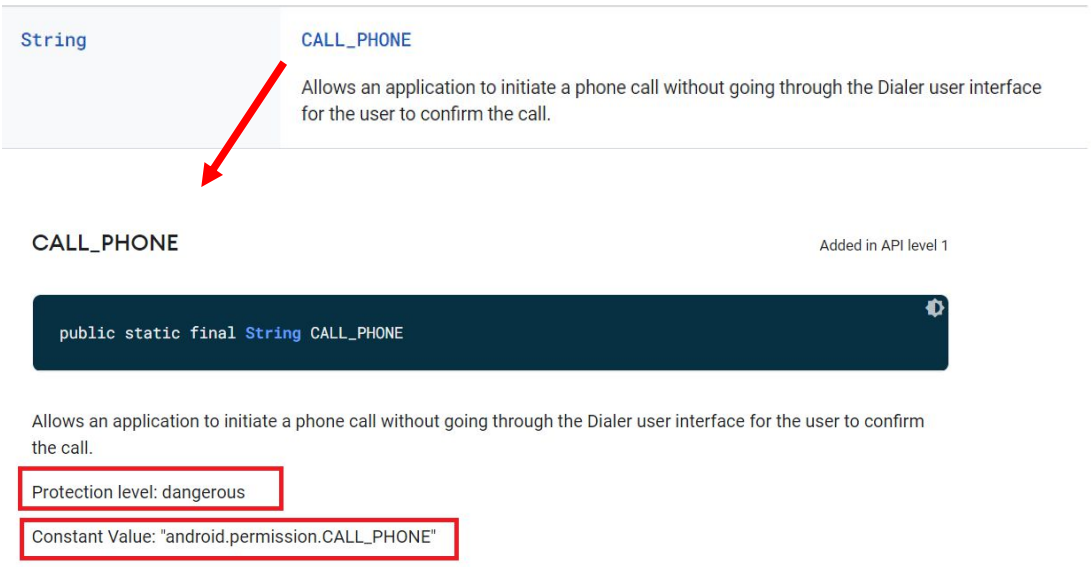
    En este caso, el sistema solicita el permiso al usuario, quien debe autorizar explícitamente a nuestra app.

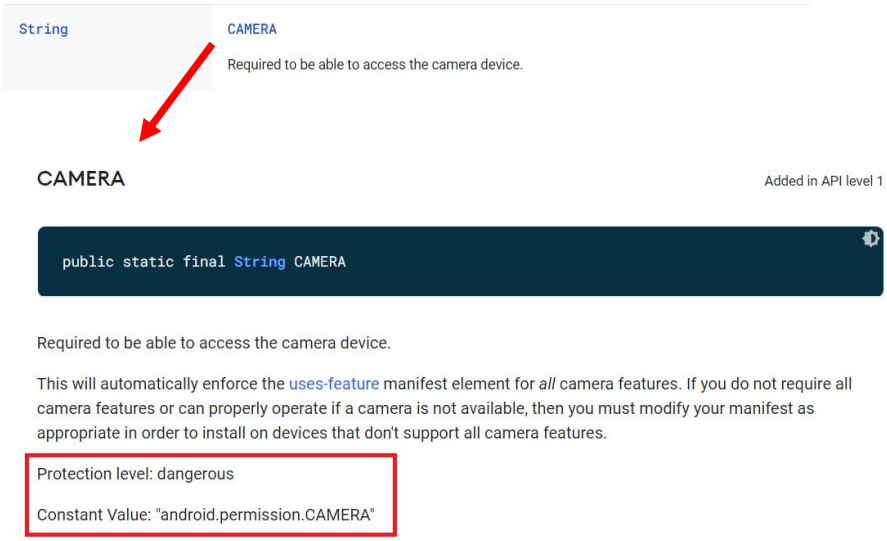
    Un ejemplo es el acceso a la cámara.

- Podemos acceder a la relación de los permisos en la dirección:

  https://developer.android.com/reference/android/Manifest.permission?hl=es

  Ejemplos:





- En versiones anteriores de la documentación se podía acceder a esta tabla donde se recogen los permisos peligrosos.

Tabla 1. Permisos riesgosos y grupos de permisos.

Grupo de permisos	Permisos
CALENDAR	• READ_CALENDAR • WRITE_CALENDAR
CAMERA	• CAMERA
CONTACTS	• READ_CONTACTS • WRITE_CONTACTS • GET_ACCOUNTS
LOCATION	• ACCESS_FINE_LOCATION • ACCESS_COARSE_LOCATION
MICROPHONE	• RECORD_AUDIO
PHONE	• READ_PHONE_STATE • CALL_PHONE (...)

- **Flujo de trabajo para el uso de permisos**

Documentación: https://developer.android.com/guide/topics/permissions/overview?hl=es-419

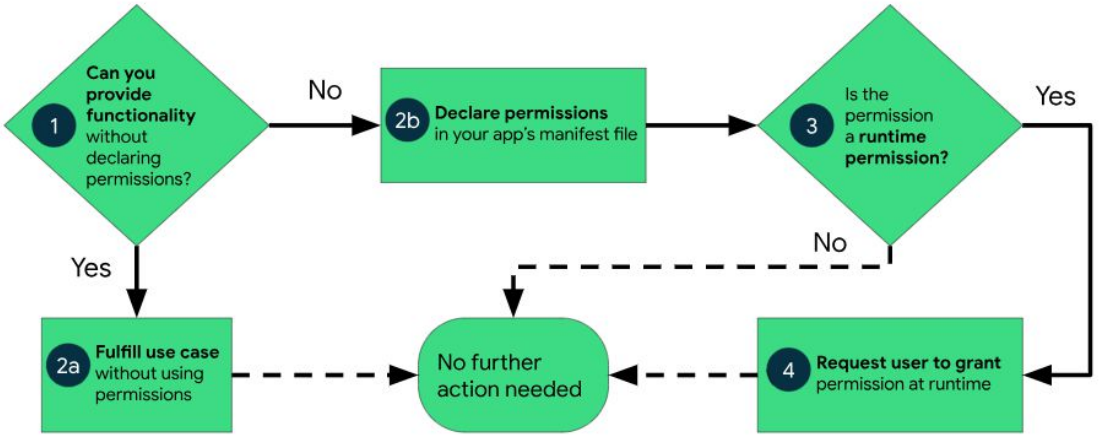
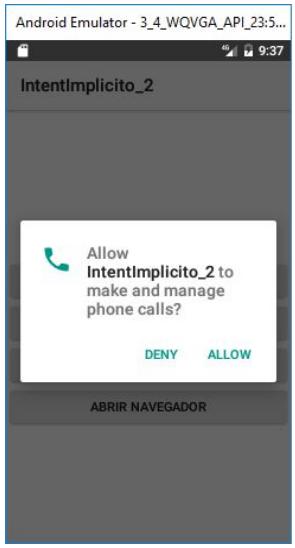


Figura 1: Flujo de trabajo general para usar permisos en Android

- Si una aplicación necesita alguno de estos permisos debemos incluir código para poder comprobar si el permiso ha sido concedido o no. Podemos hacerlo con el método ***checkSelfPermission()***:

```
if (checkSelfPermission(Manifest.permission.CALL_PHONE)==PackageManager.PERMISSION_GRANTED){
   //existe el permiso
}
else {
   //no hay permiso
}
```

- Si no ha sido concedido el permiso podemos solicitar al sistema operativo que muestre una ventana de diálogo para preguntar al usuario si acepta o deniega dicho permiso:



- Podemos hacerlo con el método ***requestPermissions()***:

```
requestPermissions(new String[]{Manifest.permission.CALL_PHONE},LLAMADA_TELEFONO);
```

El segundo parámetro es una constante entera necesaria para identificar a esta solicitud de permiso (en mi código, es ***LLAMADA_TELEFONO)***

- Como siguiente paso, necesitamos saber la elección del usuario, es decir, si ha concedido el permiso o no. Para ello, disponemos de un **método callback** que será lanzado cada vez que el sistema operativo notifique que un permiso ha sido concedido o denegado. Se trata del método ***onRequestPermissionsResult()***:

```
@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    //vemos si el código de respuesta coincide con el identificador de nuestra solicitud
    if (requestCode==LLAMADA_TELEFONO){
        //vemos si el permiso está concedido
        if(grantResults[0]==PackageManager.PERMISSION_GRANTED){
            //permiso concedido
        }
        else {
            //permiso denegado
        }
    }
}
```

- Utilización de ***<uses-feature>***:

  https://developer.android.com/guide/topics/manifest/uses-feature-element

  - Es un elemento que también se utiliza en el archivo AndroidManifest
  - Su función es informar sobre el conjunto de funciones hardware y software requeridas por la aplicación.
  - Cada función requiere su propia etiqueta ***<uses-feature>****, por lo tanto, una aplicación que requiera varias funciones debe declarar otros tantos elementos <uses-feature>*
  - Por ejemplo, una aplicación que requiera las funciones de Bluetooth y cámara en el dispositivo debe declarar estos dos elementos:

```
<uses-feature android:name="android.hardware.bluetooth" android:required="true" />
<uses-feature android:name="android.hardware.camera" android:required="true" />
```

  - Atributos:

    android:name: indica la función hardware o software usada por la app. Se indica con una cadena de caracteres.

    https://developer.android.com/guide/topics/manifest/uses-feature-element#hw-features

    android:required: especifica si la app requiere la función declarada, y no puede funcionar sin ella (valor true), o si es una preferencia el contar con dicha funcionalidad, pero la app puede funcionar igualmente sin ella (valor false).

  - Los elementos <uses-feature> declarados solo cumplen una función informativa, es decir, el sistema no comprueba que el dispositivo sea compatible con esas funciones antes de instalar una aplicación. Sin embargo, es posible que otros servicios (como Google Play) u otras aplicaciones sí puedan controlar las declaraciones <uses-feature> de una aplicación.

8. FILTROS DE INTENCION

- Se indican en el archivo AndroidManifest y **determinan la acción** que puede llevar a cabo un componente (como una Activity, en nuestro caso) y los tipos de intents que pueden gestionarla.
- Los filtros se pueden clasificar por acción, por categoría o por tipo de datos.
- **No es necesario que esté declarado un intent filter cuando se invoca a una actividad directamente, mediante el nombre de su clase**. Por eso, hasta ahora no necesitamos emplearlos para llamar a nuestras activities de forma explícita.
- Pero, si deseásemos que una actividad nuestra sea capaz de responder a un determinado tipo de intent implícito, deberíamos hacer uso de un ***Intent Filter***.

- Los *Intent-Filters* son utilizados como medio para registrar actividades en el sistema como ***"capaces de realizar una determinada acción sobre unos datos concretos"***. Con un *Intent-Filter* se anuncia al resto del sistema que nuestra aplicación puede responder a peticiones de otras aplicaciones instaladas en el dispositivo.
- Cuando una aplicación se instala en un dispositivo, el sistema operativo identifica sus filtros de intención y añade la información a un ***catálogo interno con todas las intents que soportan las aplicaciones instaladas***.

  Cuando una Activity de una aplicación llama al método ***startActivity()*** o ***startActivityForResult()***, con una intención implícita, **el sistema comprueba qué actividad (o actividades) pueden responder a la intención**.
- Si un componente, como una Activity, no declara filtros de intención sólo podrá atender a llamadas explícitas (cosa que nos ocurrió hasta ahora con nuestras activities → no declaraban ningún filtro de intención → sólo eran llamadas de forma explícita).
- Los filtros se indican en el AndroidManifest mediante la etiqueta **<intent-filter>**, dentro de la Activity correspondiente.
- El filtro de intención que crea por defecto el sistema para la actividad principal de un proyecto recién creado es:

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />

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

- Para que una Activity pueda ser llamada de modo implícito debe contener un intent-filter con una acción, y el valor "**android.intent.category.DEFAULT**" en el elemento <category>.

  De hecho, los métodos ***startActivity()*** y ***startActivityForResult()*** tratan a todos los intents como si tuviesen declarada la categoría CATEGORY_DEFAULT
- Cada elemento intent-filter puede contener múltiples elementos <category>.
- Las **categorías estándar** están definidas en la **clase Intent** como **constantes**. El nombre de dichas constantes es de la forma *CATEGORY_nombre*.
- Podemos ver una captura obtenida de la documentación en la página siguiente

  Documentación: https://developer.android.com/reference/android/content/Intent#standard-categories
- En el elemento ***<category>***, el string que contiene el valor asignado al atributo **android:name** se forma con el prefijo "android.intent.category" seguido del nombre que sigue a la palabra CATEGORY_ en la declaración de categorías disponibles. Por ejemplo, para la categoría ***CATEGORY_LAUNCHER***, el valor del string es **android:name="android.intent.category.LAUNCHER"**.

Standard Categories

These are the current standard categories that can be used to further clarify an Intent via `addCategory(String)`.

- `CATEGORY_DEFAULT`
- `CATEGORY_BROWSABLE`
- `CATEGORY_TAB`
- `CATEGORY_ALTERNATIVE`
- `CATEGORY_SELECTED_ALTERNATIVE`
- `CATEGORY_LAUNCHER`
- `CATEGORY_INFO`
- `CATEGORY_HOME`
- `CATEGORY_PREFERENCE`
- `CATEGORY_TEST`
- `CATEGORY_CAR_DOCK`
- `CATEGORY_DESK_DOCK`
- `CATEGORY_LE_DESK_DOCK`
- `CATEGORY_HE_DESK_DOCK`

(. . .)

9. GUARDAR EL ESTADO DE UNA ACTIVIDAD

- Cuando el usuario ha estado utilizando una actividad y, tras cambiar a otras, regresa a la primera, lo habitual es que ésta permanezca en memoria y continúe su ejecución sin alteraciones. Sin embargo, en situaciones de **escasez de memoria**, es posible que el sistema haya eliminado el proceso que ejecutaba la actividad. En este caso, el proceso será creado de nuevo, pero puede ocurrir que se haya perdido su estado, es decir, que se haya perdido, por ejemplo, el valor de sus variables.

- También se presenta un problema similar cuando se produce un **cambio de configuración en tiempo de ejecución**: el sistema destruye la Activity actual y la crea de nuevo para aprovechar los recursos que se adapten mejor a esta nueva configuración. Esto sucede, por ejemplo, cuando se cambia la orientación del terminal.

- Por defecto, el sistema Android usa un objeto de tipo **Bundle** para guardar información acerca de cada objeto (**View**) de la interfaz de usuario de una actividad (como, por ejemplo, el texto introducido en un objeto EditText). Así que, si la instancia de una actividad se destruye y se vuelve a crear, el estado de la interfaz se restaura al estado previo de forma automática. Sin embargo, pueden existir otros datos que no van a ser guardados automáticamente.

- Como ya se había indicado anteriormente en este mismo documento, para que el sistema pueda restaurar el estado de las vistas de una actividad, **cada vista debe tener un ID único**, indicado mediante el atributo ***android:id***.

- Existen diferentes formas de guardar datos adicionales sobre el estado de la actividad. Una de estas formas es sobreescribiendo el método ***onSaveInstanceState():***

  protected void onSaveInstanceState(Bundle estado)

- La implementación por defecto del método ***onSaveInstanceState(Bundle estado)*** hace que todas las vistas de la Activity guarden su estado como datos en el objeto Bundle. Este objeto bundle es el que se pasa como parámetro a onCreate():

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
     (…)
```

  En la llamada a la superclase de la activity se recuperan los estados guardados de las vistas y se utilizan para volver a crear la jerarquía de vistas de la Activity.

- Podemos aprovechar dicho objeto Bundle para guardar parejas “clave=valor” con los datos que deseemos almacenar (datos de los tipos básicos, como int, float, boolean, objetos String, o bien objetos de clases que implementen las interfaces ***Serializable*** o ***Parcelable***).

- El sistema pasará este objeto Bundle al evento ***onCreate()*** y también a un método llamado ***onRestoreInstanceState(),*** que se ejecuta después del método ***onStart()***. Es decir, tanto el método onCreate() como onRestoreInstanceState() reciben el mismo Bundle que contiene la información de estado. De este modo, la próxima vez que la actividad sea creada, se pueden usar estos datos para restablecer el estado que tenían las variables antes de que la actividad fuese destruida.

- En resumen:

  - ***onSaveInstanceState(Bundle)****:* Se invoca para permitir a la actividad guardar su estado.
  - ***onRestoreInstanceState(Bundle)***: Se invoca para recuperar el estado guardado por ***onSaveInstanceState()***.

- Los métodos ***onSaveInstanceState()*** y ***onRestoreInstanceState()*** no forman parte del ciclo de vida de una actividad y, por tanto, no son llamados siempre que una actividad es destruida (por ejemplo, cuando el usuario pulsa la tecla de retroceso o cuando se ejecuta el método ***finish()***).

- La ventaja de usar estos métodos es que el programador no ha de buscar un método de almacenamiento permanente, es el sistema quien hará este trabajo.

- Ejemplo: guardar la información de una variable de tipo cadena:

```
String var;

@Override
protected void onSaveInstanceState(Bundle guardarEstado) {
    super.onSaveInstanceState(guardarEstado);
    guardarEstado.putString("variable", var);
}

@Override
protected void onRestoreInstanceState(Bundle recEstado) {
    super.onRestoreInstanceState(recEstado);
    var = recEstado.getString("variable");
}
```

- Siempre hay que llamar a la superclase para que la implementación por defecto pueda guardar/restaurar el estado de las vistas.

- Como el método *onCreate()* también recibe el mismo bundle, es posible utilizar este método para recuperar los datos guardados mediante *onSaveInstanceState()*. Pero, dado que el método *onCreate()* se llama tanto si el sistema está creando una nueva instancia de la actividad como si está recreando una previa, debemos comprobar si el Bundle de estado es nulo antes de intentar leerlo. Si es nulo, significa que el sistema está creando una nueva instancia de la actividad, en lugar de restaurar una anterior que había sido destruida:

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Comprobamos si estamos recreando una instancia destruida previamente
    if (savedInstanceState != null) {
        // Restauramos los valores del estado guardado
        var = savedInstanceState.getString("variable");
    } else {
        // Sentencias…
    }
}
```

- En el caso de usar *onRestoreInstanceState()*, el sistema sólo llama a este método si hay un estado guardado que restaurar, por lo que no es necesario comprobar si el bundle es nulo.

- Otra solución para evitar la posible pérdida de datos cuando el terminal cambia de orientación es bloquear nuestra aplicación para que no pueda cambiar de orientación con el terminal y, por tanto, no sea destruida y creada de nuevo. Esto se puede conseguir de dos formas:

  - **En el archivo AndroidManifest**, dentro de la actividad, mediante una línea de código como las siguientes:

    ```
    android:screenOrientation="portrait"
    android:screenOrientation="landscape"
    ```

  - **Mediante código Java**:

    ```
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
    ```