ACTIVIDADES

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

1. INTRODUCCION

- Una **Activity** (Actividad) es un elemento de Android que muestra una pantalla con elementos (vistas: botones, texto, imágenes, etc.) con los que los usuarios pueden interactuar.
- Una Activity, por lo general, ocupa toda la pantalla.
- 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.
- Cada aplicación puede tener múltiples actividades. Todas las actividades que componen una aplicación deben estar registradas en el archivo **AndroidManifest**.

2. ARCHIVO AndroidManifest

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

```
<?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>
```

- La etiqueta raíz **<manifest>** contiene dos atributos: declaración del espacio de nombres y nombre del paquete que contiene la aplicación.

- 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

- Desde una actividad de una aplicación se puede saltar a otra actividad de otra aplicación (por ejemplo, desde WhatsApp se puede saltar a consultar los datos de un contacto de la aplicación de contactos).

- Android organiza las actividades en una pila de ejecución LIFO ("last in, first out" o "último en entrar, primero en salir").

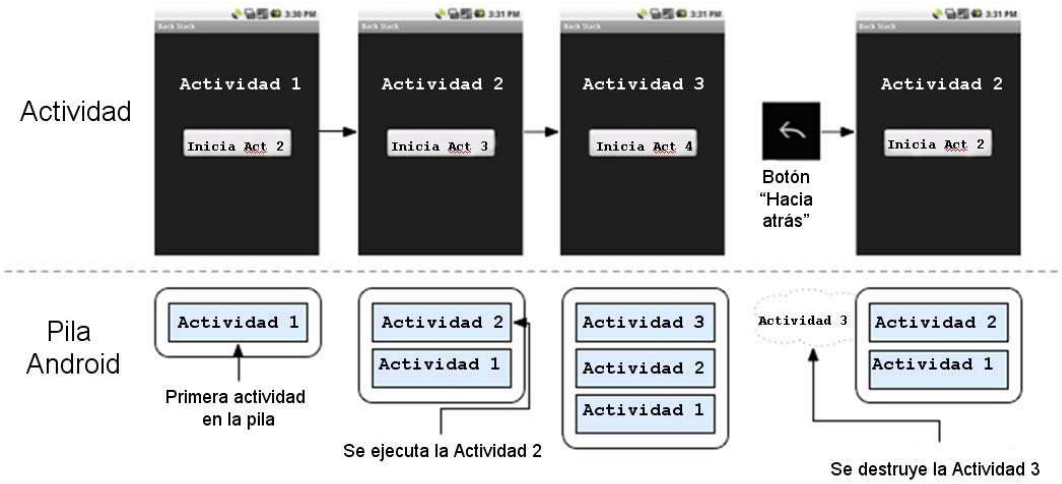
- La pila no solamente contiene las Activity de nuestra aplicación sino que todas las aplicaciones comparten dicha pila.

- 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.

- Al pulsar la tecla de retroceso del móvil, se destruye la actividad actual y se vuelve a cargar la actividad que se encuentra en la parte superior de la pila.

- El siguiente esquema representa cómo cambia la pila de Android al ir abriendo actividades y al pulsar el botón "Volver Atrás":



- Desde que iniciamos una actividad hasta que salimos de ella, dicha actividad pasa por diferentes etapas, y esto es lo que se conoce como **ciclo de vida de una actividad**.

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:

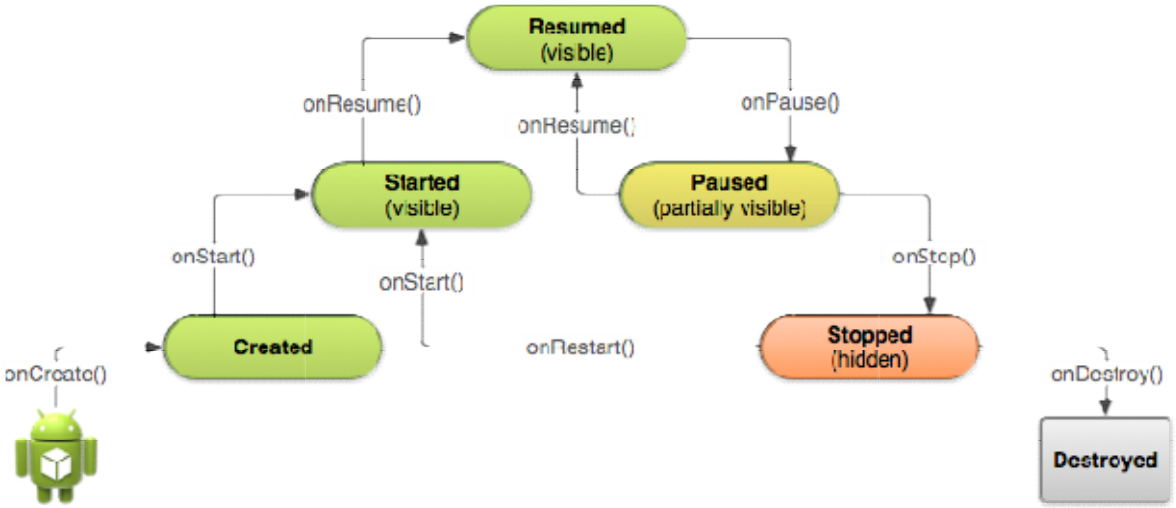
    - **Resumed**: la actividad está en el primer plano de la pantalla (**visible**): **ejecutándose**.

    - **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. Por 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 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 actividad (pantalla), de la misma aplicación o de otra distinta.
  - Pasar datos de una actividad a otra, de su misma aplicación o de otra distinta.
  - Interconectar otros componentes de la misma o distinta aplicación.
- Las intenciones 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 intención (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:

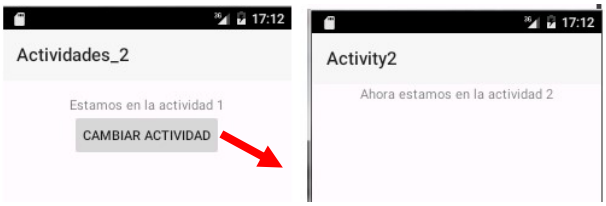
```
startActivityForResult(unIntent, INTENT_COD);
```

  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.

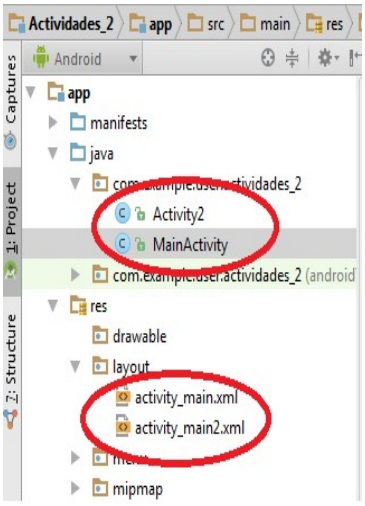
- El fichero "**AndroidManifest**" debe incluir todas las actividades (y demás componentes de una aplicación), ya que, si no lo hacemos, no son visibles para el sistema y, en consecuencia, no se pueden ejecutar. La actividad principal ya debe aparecer puesto que se creó de forma automática al crear el 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 Actividades_2.



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.user.actividades_2;

    (…)

public class MainActivity extends Activity {

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

        btn = (Button)  findViewById(R.id.btnCambiarActividad);
    }

    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

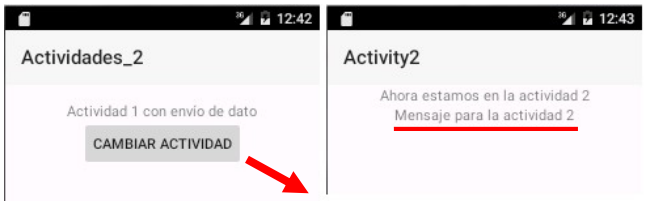
```
package com.example.user.actividades_2;

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

public class Activity2 extends Activity {

    @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 Actividades_2, de forma que al llamar a la segunda actividad le envíe un dato, por ejemplo, la cadena de caracteres "Mensaje para la actividad 2", como muestran las siguientes capturas.



Código de la Activity principal

```
package com.example.user.actividades_2;

(…)

public class MainActivity extends Activity {

    private Button btn;
    private String datoEnviado = "Mensaje para la actividad 2";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        btn = (Button)  findViewById(R.id.btnCambiarActividad);
    }

    public void onClickCambiarActividad (View v){
        Intent miIntent = new Intent(this, Activity2.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.user.actividades_2;

(…)

        public class Activity2 extends Activity {

            private TextView lbl2;
            private String datoRecibido;
            @Override
            protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main2);

                lbl2 = (TextView) findViewById(R.id.lbl2);

                Intent intent = getIntent();
                datoRecibido = intent.getExtras().getString("dato");
                lbl2.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.

En segundo lugar, hemos recuperado la información asociada mediante el método ***getExtras()***, de la clase Intent.

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

Y con el método ***getString()*** obtenemos el texto que se ha pasado como parámetro desde la Activity que ha iniciado el lanzamiento de esta segunda actividad. Y dicho parámetro estaba bajo la clave "*dato*":

```
Intent intent = getIntent();
datoRecibido = intent.getExtras().getString("dato");
```

Para recuperar el valor del dato extra también podríamos haber utilizado el método ***getStringExtra()***, de la clase Intent:

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

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

Por último, construimos el texto de la etiqueta a mostrar.

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

```
(…)
    public void onClickCambiarActividad (View v){
        Intent miIntent = new Intent(this, Activity2.class);
        // en este caso vamos a pasar un objeto de tipo Bundle:
        // primero lo definimos y luego le añadimos el String
        Bundle unBundle = new Bundle();
        unBundle.putString("dato", datoEnviado);
        // asociamos el objeto Bundle al intent
        miIntent.putExtras(unBundle);
        startActivity(miIntent);
    }
}
```

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

```
        lbl2 = (TextView) findViewById(R.id.lbl2);

        // recupera el objeto Bundle que se le ha pasado,
        // mediante el método getExtras() del objeto Intent
        Bundle unBundle = getIntent().getExtras();
        // obtiene el dato desde el Bundle mediante el método
getString()
        datoRecibido = unBundle.getString("dato");
        lbl2.setText(datoRecibido);
    }
}
```

Comentario:

Para recuperar los datos que se han enviado a través del objeto Bundle, utilizamos el método ***getExtras()*** del objeto Intent. Este método devuelve un objeto Bundle que podemos utilizar para recuperar con el método get<type>, diferentes pares "clave-valor", cuyos tipos pueden ser String, int, etc. (Es decir, get<type> sería concretamente un método ***getString()***, ***getInt()***, etc.).

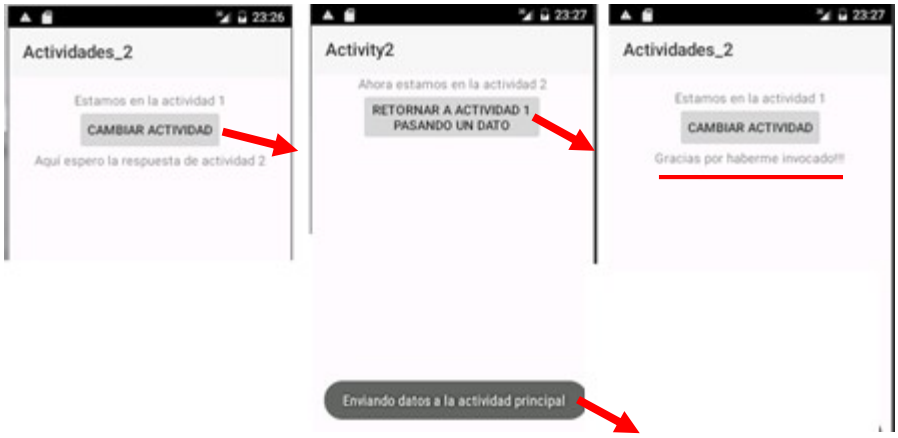
6.3 Lanzar una segunda actividad de la misma aplicación esperando respuesta.

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 realiza 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 trabajar con una copia del proyecto anterior (**Actividades_2)**. Consistirá en que la actividad 1 invoque a la actividad 2 con intención de que ésta le retorne un dato. Y la actividad2 va a retornar una cadena de caracteres a la actividad1.



<u>Archivo de layout para la pantalla inicial</u> (activity_main.xml)

Añadimos una TextView donde recuperaremos el dato retornado.

<u>Archivo de layout para la segunda pantalla</u> (activity_main2.xml)

Añadimos un Button para enviar dato desde la Activity2.

<u>Código de la Activity principal</u>

```
public class MainActivity extends Activity {
    (...)
    private static final int CODIGO=1;

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

    @Override
    protected void onActivityResult (int requestCode, int resultCode, Intent
datos){
        // comprueba si el código devuelto es el código de la solicitud
        if (requestCode == CODIGO) {
            // comprueba si el codigo del resultado es OK
            if (resultCode == RESULT_OK){
                // recupera el resultado
                datoRecibido = datos.getExtras().getString("dato");
                lbl2.setText(datoRecibido);
            }
        }
    } // end onActivityResult

}
```

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, y es el mismo que nos va a devolver

la actividad hija cuando finalice su ejecución. Su función es identificar la actividad que envía 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 class Activity2 extends Activity {

    private String datoARetornar = "Gracias por haberme invocado!!!";
    (…)

    public void onClickRetornarDato (View v){
        // devolvemos el dato a la actividad que realiza la llamada
        Bundle b = new Bundle();
        b.putString("dato", datoARetornar);
        // asociamos el objeto Bundle al objeto Intent utilizado para
devolver datos
        Intent i = new Intent();
        i.putExtras(b);
        // enviamos los datos
        setResult(RESULT_OK, i);
        Toast.makeText(this, "Enviando datos a la actividad principal ",
                       Toast.LENGTH_SHORT).show();
        // finalizamos la ejecucion
        finish();
    }// end onClickRetornaApellido
}
```

Comentario:

Para devolver el resultado (en este ejemplo, no es un resultado en sí, sino que se trata de una cadena de caracteres definida en la activity) a la actividad que realiza la llamada, se utiliza el método ***setResult()***, con dos parámetros.

El primer parámetro casi siempre será 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***. Por ejemplo, si una activity hija tiene un botón OK y otro Cancel, debería establecer un código ***resultCode*** diferente según qué botón haya sido pulsado. De esta forma, la actividad padre podrá continuar con acciones distintas dependiendo de cada código de resultado.

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

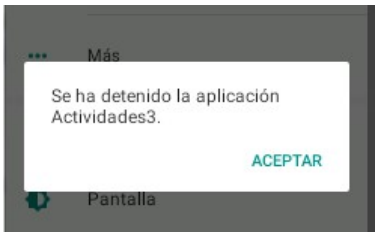
6.4 Lanzar una actividad de otra aplicación

- En Android, las actividades pueden ser invocadas por cualquier aplicación que se ejecute en el dispositivo.
  Así, podemos hacer que una actividad lance otra actividad perteneciente a otra aplicación, como se ve en el código siguiente:

```
Intent i = new Intent();
i.setClassName("com.example.user.aplicacionX","com.example.user.aplicacionX.Actividad_llamada");
```

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**.

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



- Para verificar que haya una actividad disponible que pueda responder al intent podemos hacer 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();
}
```

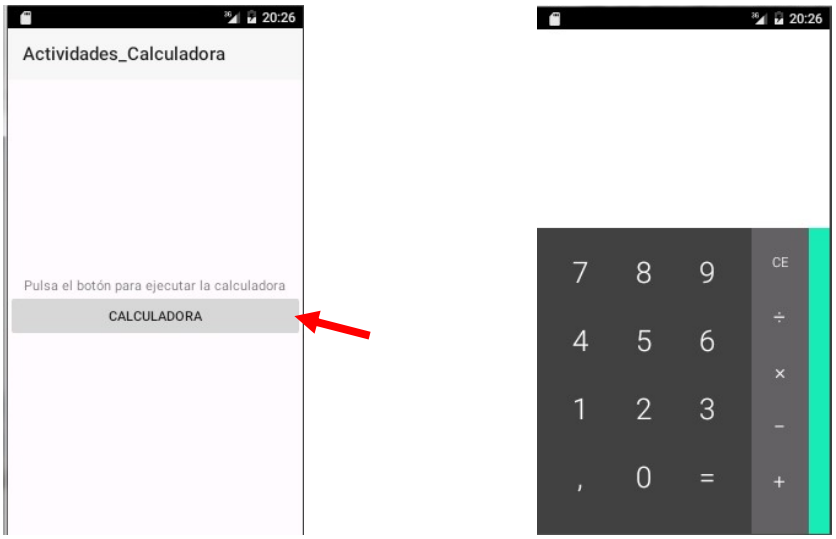
- El ***PackageManager*** "conoce" todos los componentes instalados en el dispositivo, incluidas todas las activity. Creamos una instancia de la clase PackageManager mediante ***getPackageManager()***.
- Invocando ***queryIntentActivities(Intent, int)*** obtenemos una lista de actividades que se corresponden 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)).

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

6.5 Lanzar una actividad de una aplicación del sistema

Una aplicación puede llamar a las diferentes aplicaciones incorporadas en el dispositivo Android.

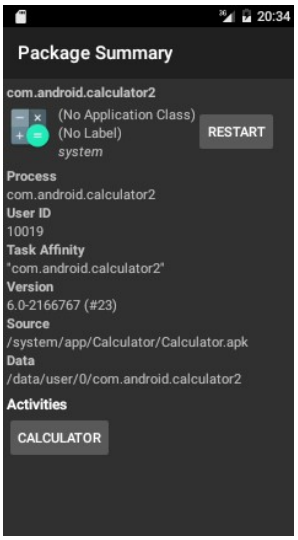
Por ejemplo, vamos a realizar un proyecto llamado "**Actividades_Calculadora**", que lance la aplicación de la calculadora como respuesta a la pulsación de un botón, como se ve en las imágenes siguientes:



El código necesario para esto sería:

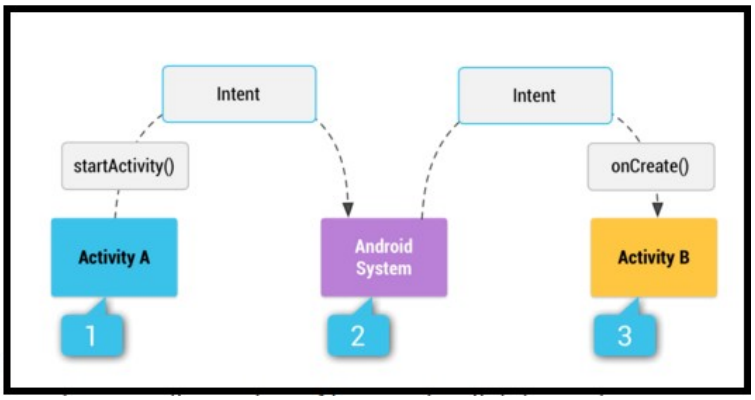
```
Intent i = new Intent();
//el método setClassName(String packageName, String className) indica el nombre del paquete que
// contiene la clase y el nombre de dicha clase.
i.setClassName("com.android.calculator2", "com.android.calculator2.Calculator");
startActivity(i);
```

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



6.6 Intents implícitos.

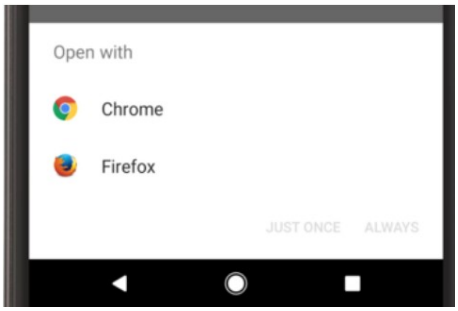
- Un componente Android, como una actividad, puede ser lanzado de forma implícita cuando se indica la **acción** que se desea realizar y, si es el caso, los **datos** sobre los que se va a realizar dicha acción.
- A diferencia de los intents explícitos, no se especifica el nombre de la actividad que va a atender al intent.
- El sistema escoge la actividad que va a atender a la llamada según sea la acción indicada en el intent.



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.
  - 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.

- Para lanzar un Intent implícito se indicará:
  - La **acción** que se desea llevar a cabo. 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.
  - Los **datos** sobre los que ejecutar dicha acción, por ejemplo, una URL, los datos de un contacto del teléfono, etc.

- En **developer.android.com/guide/appendix/g-app-intents.html** se puede encontrar una lista con los intents que nuestra aplicación puede enviar para invocar algunas aplicaciones de un dispositivo Android

- También se recogen las diferentes “action” en http://developer.android.com/intl/es/reference/android/content/Intent.html

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`

(...)

- Ejemplo: Llamar a un determinado número de teléfono:
  - Acción: Intent.ACTION_CALL
  - Dato: un objeto Uri, cuyo contenido sea el número de teléfono
  - El intent implícito quedaría:

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

7. PERMISOS

Documentación: http://developer.android.com/reference/android/Manifest.permission.html

- 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 acceso a ésta, pero no a la ubicación del dispositivo. El usuario puede revocar los permisos en cualquier momento desde la pantalla de configuración de la app.

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

  - Los permisos del sistema se dividen en dos categorías, "*normal*" y "*peligroso*":
    - Los permisos **normales** no ponen en riesgo la privacidad del usuario de forma directa. Si una app tiene un permiso normal en su manifiesto, el sistema concede el permiso automáticamente.
    - Los permisos **peligrosos** pueden permitir que la app acceda a información confidencial del usuario. Si tienes un permiso peligroso, el usuario debe autorizar explícitamente a tu app.
  - Podemos acceder a la relación de los permisos considerados críticos en la dirección:

    https://developer.android.com/guide/topics/security/permissions.html?hl=es#normal-dangerous
  - También desde esa dirección podemos enlazar con la página que contiene la relación de permisos considerados normales.

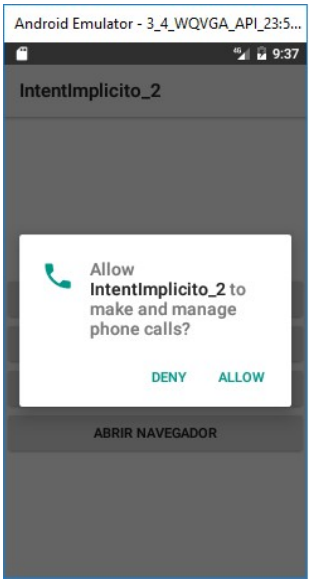
Tabla 1. Permisos riesgosos y grupos de permisos.

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 (...)

- 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.

- 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
        }
    }
}
```

- Google ha elaborado una guía de buenas prácticas a la hora de gestionar este tipo de permisos. Podemos acceder a ella desde

https://developer.android.com/training/permissions/best-practices.html?hl=es

8. FILTROS DE INTENCION

- Si deseamos que una actividad nuestra sea capaz de responder a un determinado tipo de intent implícito, deberemos 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 identifica sus filtros de intención y añade la información a un catálogo interno con todas las intents que soportan las aplicaciones instaladas. Cuando una aplicación llama a ***startActivity()*** o ***startActivityForResult()***, con una intención implícita, el sistema comprueba qué actividad (o actividades) pueden responder a la intención.
- Si un elemento, como una Activity, no declara filtros de intención sólo podrá atender a llamadas explícitas.
- Los filtros se indican en el AndroidManifest mediante la etiqueta **<intent-filter>**, dentro de la actividad 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>
```

- Todas las actividades que puedan ser llamadas de modo implícito deben contener un intent-filter con una acción, y el valor "**android.intent.category.DEFAULT**" en el elemento <category>. 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*:

  - 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
  - CATEGORY_CAR_MODE
  - CATEGORY_APP_MARKET

  (...)

- El string que contiene el valor asignado al atributo **android:name** se forma anteponiendo el prefijo "android.intent.category" al nombre que sigue a la palabra CATEGORY_. Por ejemplo, para la categoría CATEGORY_LAUNCHER, el valor sería **android:name="android.intent.category.LAUNCHER"**.

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

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:

```
@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);
        ```