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 ***Activitys*** (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.

- 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>
```

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

- 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:

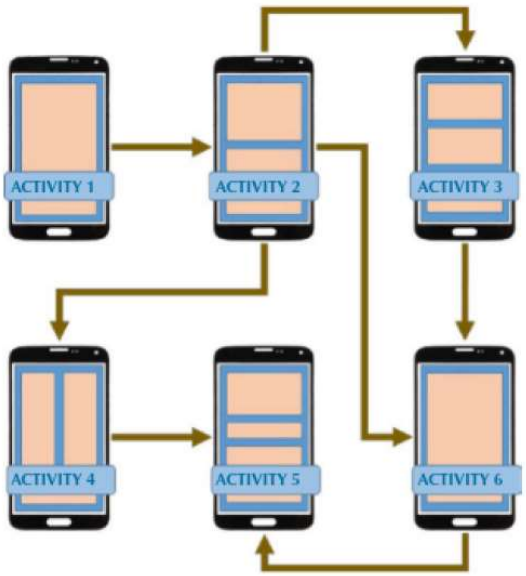
    - *"**android.intent.action.MAIN**"* sirve para que el sistema operativo sepa qué actividad lanzar en primer lugar.

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

3. PILA DE ACTIVIDADES

- Una aplicación generalmente consta de múltiples actividades vinculadas entre sí. Las Activities (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.

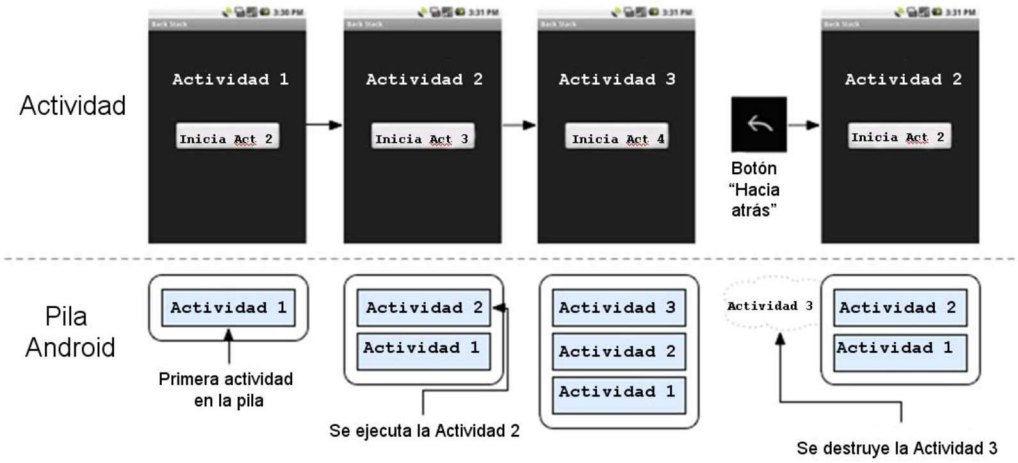


(imagen obtenida de **CABRERA RODRIGUEZ: "Programación multimedia y dispositivos móviles". Ed. Síntesis**)

- 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).

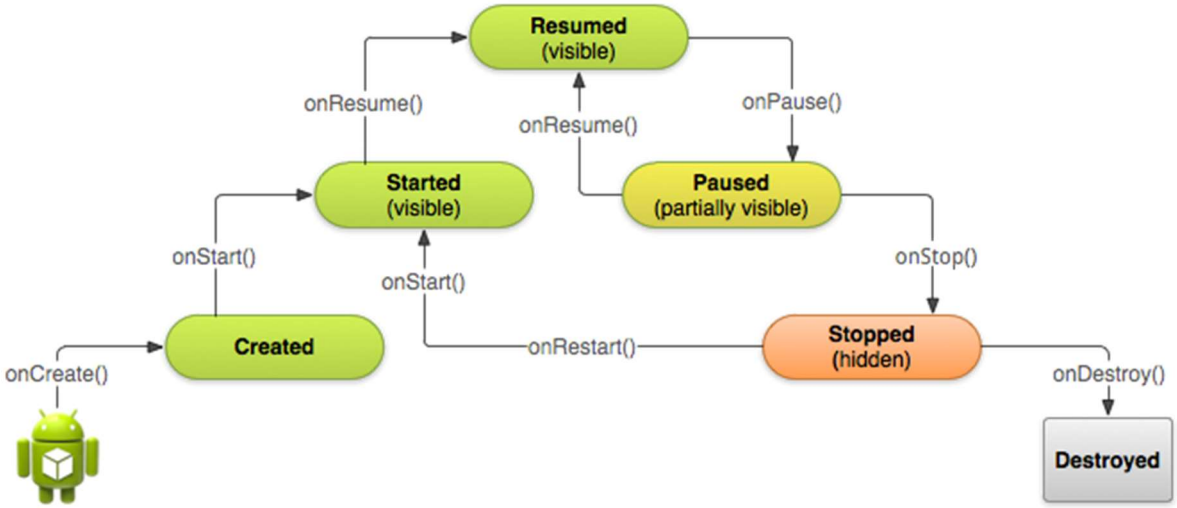
- Desde que iniciamos una actividad hasta que salimos de ella, dicha actividad pasa por diferentes etapas o estados. Para Android el estado de la actividad puede ser:

  - *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 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 pque 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.

- 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 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:

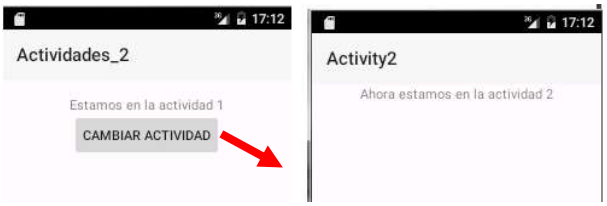
```
startActivityForResult(unIntent, INTENT_COD);
```

  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.

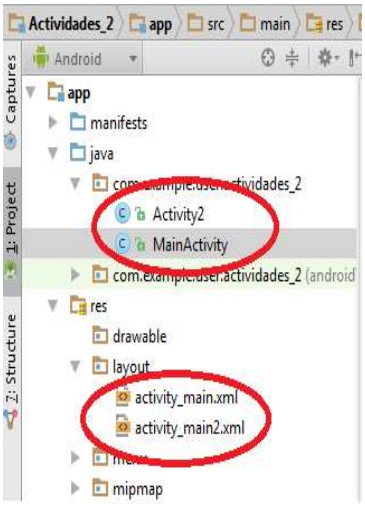
- El fichero "**AndroidManifest**" debe incluir todas las actividades (y demás componentes de una aplicación), ya que, si no lo hacemos, no son visibles para el sistema y, en consecuencia, no se pueden ejecutar. La actividad principal ya debe aparecer puesto que se creó de forma automática al crear el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

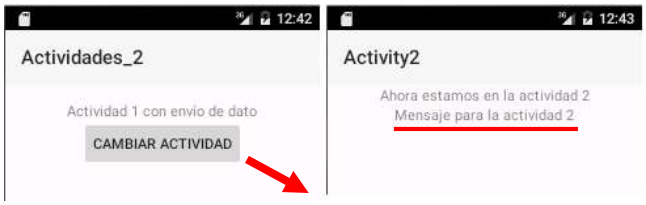
```
package com.example.user.actividades_2;

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 = 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.

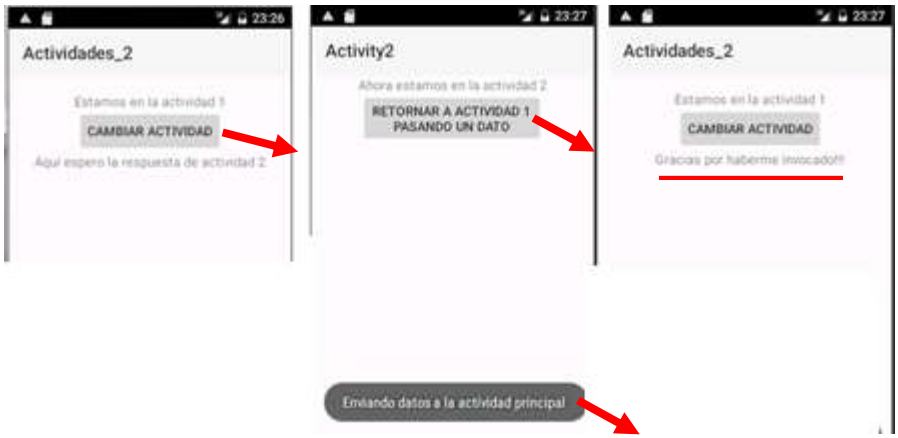
Documentación: http://developer.android.com/training/basics/intents/result.html

- 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()***.