Intents y navegación entre actividades

Índice

En esta segunda sesión hablaremos en primer lugar de un concepto muy importante en el desarrollo de aplicaciones Android: el de `Intent`. Veremos que entre sus múlitples usos se encuentra el de iniciar nuevas actividades, por lo que también le prestaremos especial atención a cómo se puede gestionar la forma en la que se navega entre las diferentes actividades de una aplicación, ampliando la información que se dio en la sesión anterior sobre este tema.

1. Intents

Los `Intents` son uno de los elementos más característico en el desarrollo de aplicaciones para Android. Los `Intents` permiten intercambiar datos entre aplicaciones o componentes de aplicaciones, como por ejemplo las actividades. También pueden ser usados para iniciar actividades o servicios. Otra posible aplicación de los `Intents` es solicitar al sistema que se realice una determinada acción con ciertos datos; el propio Android se encargará de buscar la aplicación más cualificada para realizar el trabajo.

Por lo tanto, los `Intents` se convierten un mecanismo para la transmisión de mensajes que puede ser utilizado tanto en el seno de una única aplicación como para comunicar aplicaciones entre sí. Los posibles usos de los `Intents` son:

- Solicitar que se inicie una actividad o servicio para llevar a cabo una determinada acción, pudiéndose añadir o no datos a la solicitud. Se trata de una solicitud implícita (no especificamos la actividad o servicio a iniciar, sino que la tarea que queremos que se lleve a cabo)
- Anunciar al resto del sistema que se ha producido un determinado evento (como cambios en la conexión a Internet o el nivel de carga de la batería), de tal forma que las actividades que estén preparadas para reaccionar ante determinado evento puedan realizar una determinada operación.
- Iniciar un servicio o actividad de manera explícita.

El uso de `Intents` es un principio fundamental en el desarrollo de aplicaciones para Android. Permite el desacoplamiento de componentes de la aplicación, de tal forma que cualquiera de ellos pueda ser sustituido fácilmente. También permite de manera simple extender la funcionalidad de nuestras aplicaciones, reutilizando actividades presentes en aplicaciones de terceros o incluso aplicaciones nativas de Android.

1.1. Usar Intents para lanzar actividades

Los `Intents` pueden ser utilizados para iniciar o navegar entre actividades. Para iniciar una actividad debemos hacer una llamada a la función `startActivity`, pasando como parámetro un `Intent`:

```
startActivity(intent);
```

El `Intent` puede o bien especificar de manera explícita el nombre correspondiente a la

clase de la actividad que queremos iniciar, o bien indicar una acción que queremos que sea resuelta por una actvidad, sin especificar cuál. En este segundo caso el sistema escogerá la actividad a ejecutar de manera dinámica en tiempo de ejecución, buscando aquella que permita llevar a cabo la tarea solicitada de la manera más apropiada. Nuestra actividad no obtendrá ninguna notificación cuando la actividad recién iniciada finalice. Para ello deberemos utilizar otra función que veremos en más detalle más adelante.

Veamos en primer lugar como iniciar una Actividad de manera **explícita**. Para ello debemos crear un nuevo `Intent` al que se le pase como parámetro el contexto de la aplicción y la clase de la actividad a ejecutar:

```
Intent intent = new Intent(MiActividad.this, MiOtraActividad.class);
startActivity(intent);
```

Una llamada a la función `finish` (sin parámetros) en la nueva actividad o la pulsación del botón *BACK* de nuestro dispositivo hará que la nueva actividad se cierre y se elimine de la pila de actividades.

La otra forma de iniciar una actividad es mediante un `Intent` **implícito**, en el que se solicita que un componente anónimo de una aplicación sin determinar se encargue de satisfacer una petición concreta. Esto básicamente significa que somos capaces de solicitar al sistema que se inicie una actividad o servicio para realizar una acción sin necesidad de conocer previamente qué actividad, o incluso aplicación, se encargará de ello. Para crear un `Intent` implícito indicamos la acción a realizar y, opcionalmente, la URI de los datos sobre los que queremos que se lleve a cabo la acción. También es posible añadir datos adicionales al `Intent`, conocidos como *extras*.

Cuando se utiliza este método será el propio sistema el que, en tiempo de ejecución, decidirá qué actividad es la más adecuada para realizar la acción solicitada sobre los datos suministrados. Esto podría incluso permitir que nuestra aplicación use funcionalidades de otras, sin saber exactamente qué otra aplicación es la que nos está ayudando. Por ejemplo, para que se puedan hacer llamadas de teléfono desde nuestra aplicación podríamos o bien implementar una nueva actividad para ello, o utilizar un `Intent` implícito que solicite que se lleve a cabo la tarea de realizar la llamada a un número de teléfono representado por una URI:

```
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:666666666");
startActivity(intent);
```

En el caso en el que existan varias actividades igualmente capaces de llevar a cabo la tarea se le permitirá escoger al usuario cuál de ellas utilizar por medio de una ventana de diálogo. Los componentes y aplicaciones nativos de Android tienen la misma prioridad que el resto, por lo que pueden ser completamente reemplazados por nuevas actividades qeu declaren que pueden llevar a cabo las mismas acciones.

1.2. Obtener información de subactividades

Una actividad iniciada por medio de la función `startActivity` es independiente de su

actividad padre y por lo tanto no propocionará ningún tipo de información a ésta cuando finalice. En Android otra posibilidad es iniciar una actividad como una *subactividad* que esté conectada a su actividad padre. Una subactividad que finalice provocará siempre la activación de un evento en su actividad padre.

Nota:
Una subactividad no es más que una actividad que se ha iniciado de manera diferente. Cualquier actividad registrada en el *Manifest* de la aplicación puede ser iniciada como una subactividad.

Para lanzar una subactividad usaremos el método `startActivityForResult`, que funciona de manera muy parecida a `startActivity`, pero con una diferencia muy importante. Además de pasarle un `Intent` con una petición explícita o implícita de inicio de actividad, se le pasará también como parámetro un *código de petición*. Este código será un valor entero que será utilizado más tarde para identificar cuál de las subtareas es la que ha finalizado:

```
private static final int CODIGO_ACTIVIDAD = 1;

Intent intent = new Intent(this, MiOtraActividad.class);
startActivityForResult(intent, CODIGO_ACTIVIDAD);
```

Cuando la subactividad esté preparada para terminar, llamaremos a `setResult` antes de la llamada `finish` para devolver un resultado a la actividad padre. El método `setResult` requiere dos parámetros: el código de resultado y el propio resultado, que se representará mediante un `Intent`. Generalmente el código de resultado tendrá el valor `Activity.RESULT_OK` o `Activity.RESULT_CANCELED`, aunque podríamos utilizar nuestros propios códigos de resultado, ya que este primer parámetro puede tomar como valor cualquier número entero. El `Intent` devuelto a la clase padre contiene normalmente una URI que hace referencia a una determinada pieza de información y una colección de elementos extra usados para devolver información adicional. Veamos un ejemplo:

```
botonOk.setOnClickListener(new View.onClickListener() {
        public void onClick(View view) {
                Uri dato = Uri.parse("content://contactos/" +
id_contacto_seleccionado);

                Intent resultado = new Intent(null, dato);
                resultado.putExtra(DATOS_CORRECTOS, datosCorrectos);
                resultado.putExtra(TELEFONO_SELECCIONADO,
telefonoSeleccionado);
                setResult(RESULT_OK, resultado);
                finish();
        }
});

botonCancelar.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
                setResult(RESULT_CANCELED, null);
                finish();
        }
});
```

Como se puede observar es posible que una actividad devuelva diferentes resultados a la

actividad padre desde diferentes puntos del código. En el caso de que la actividad se cierre debido a que se pulsó el botón *BACK* en el dispositivo móvil o en el caso en el que se haga una llamada a `finish` sin haberla hecho previamente a `setResult`, el código de resultado devuelto a la actividad padre será `RESULT_CANCELED` y el `Intent` devuelto contendrá el valor `null`.

El método `putExtra` de la clase `Intent` está sobrecargado. El primer parámetro será siempre una cadena que permita identificar el dato concreto, mientras que el segundo podrá ser una cadena, un carácter, un entero, un vector, etc.

Cuando una subactividad finaliza se ejecutará el manejador del evento `onActivityResult` de la clase padre. Deberemos sobrecargar este método para poder hacer algo con los datos devueltos por una subactividad. Dicho método recibirá tres parámetros, el código de petición que fue usado para lanzar la subactividad que acaba de finalizar, el código de resultado establecido por la subactividad para indicar su resultado antes de finalizar (recuerda que este valor puede ser un entero cualquiera, aunque lo más normal es que sea `Activity.RESULT_OK` o `Activity.RESULT_CANCELED`), y el `Intent` utilizado por la subactividad para empaquetar los datos a devolver. Este `Intent` puede incluir una URI que represente una determinada pieza de información y también una serie de elementos extra. A continuación se muestra un ejemplo de implementación de este método:

```
private static final int PRIMERA_ACTIVIDAD = 1;
private static final int SEGUNDA_ACTIVIDAD = 2;

@Override
public void onActivityResult(int requestCode,
                                                    int resultCode,
                                                    Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        switch(requestCode) {
                case PRIMERA_ACTIVIDAD:
                        if (resultCode == Activity.RESULT_OK) {
                                Uri dato = data.getData();
                                boolean datosCorrectos =
data.getBooleanExtra(DATOS_CORRECTOS, false);
                                String telefono =
data.getStringExtra(TELEFONO_SELECCIONADO);
                        }
                        break;
                case SEGUNDA_ACTIVIDAD:
                        if (resultCode == Activity.RESULT_OK) {
                                // Hacer algo
                        }
                        break;
        }
}
```

1.3. Responder peticiones de Intents

Como se ha comentado anteriormente, los `Intents` implícitos permiten solicitar la realización de una determinada acción, sin especificar explícitamente qué actividad

deberá encargarse de llevarla a cabo. El sistema escoge la más adecuada. Algunas de las acciones que podemos especificar se indican a continuación. Muchas de ellas serán llevadas a cabo por actividades nativas del sistema:

- `ACTION_ANSWER`: abre una actividad que maneje llamadas de teléfono entrantes.
- `ACTION_CALL`: muestra un marcador de teléfonos e inmediatamente inicia una llamada usando el número indicado en la URI del `Intent`.
- `ACTION_DELETE`: inicia una actividad que permita eliminar los datos referenciados por la URI del `Intent`.
- `ACTION_DIAL`: muestra un marcador de teléfonos, con un número premarcado (aquel pasado mediante la URI del `Intent`).
- `ACTION_EDIT`: solicita la ejecución de una actividad que permita la edición de los datos referenciados por la URI del `Intent`.
- `ACTION_INSERT`: solicita la ejecución de una actividad que permita la inserción de nuevos elementos en el `Cursor` especificado por la URI. El concepto de `Cursor` será tratado más adelante en el curso.
- `ACTION_PICK`: lanza una actividad que permita escoger un elemento del Proveedor de Contenidos especificado por la URI del `Intent`. La aplicación seleccionada dependerá del tipo de dato que se desee escoger. Por ejemplo, usando `content://contacts/people` como URI hará que se lance la lista de contactos nativa del dispositivo. El concepto de Proveedor de Contenidos será explicado más adelante durante el curso.
- `ACTION_SEARCH`: lanza una actividad que permita realizar una búsqueda. El término a buscar deberá ser incluido en el `Intent` como un extra usando la clave `SearchManager.QUERY` como clave.
- `ACTION_SENDTO`: lanza una actividad que permita enviar un mensaje al contacto especificado mediante la URI del `Intent`.
- `ACTION_SEND`: lanza una actividad que sea capaz de enviar los datos especificados mediante el `Intent`.
- `ACTION_VIEW`: es la acción más común. Se solicita que los datos referenciados por la URI del `Intent` sean visualizados de la manera más adecuada posible.
- `ACTION_WEB_SEARCH`: abre una actividad que realiza una búsqueda en Internet basada en la URI del `Intent`.

En el caso en el que deseemos que nuestra actividad 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.

Para registrar una actividad como manejador potencial de un determinado tipo de `Intent` añadimos un elemento `intent-filter` a su correspondiente nodo `activity` en el *Manifest* de la aplicación:

```
<activity android:name=".MiActividad" android:label="Mi Actividad">
```

```
        <intent-filter>
                <action
android:name="es.ua.jtech.intent.action.HAZ_ALGO"/>
                <category android:name="android.intent.category.DEFAULT"/>
                <category
android:name="android.intent.category.ALTERNATIVE_SELECTED"/>
                <data android:mimeType="vnd.miaplicacion.cursor.item/*"/>
        </intent-filter>
</activity>
```

Dentro del elemento `intent-filter` incluiremos los siguientes nodos:

- `action`: usa el atributo `android:name` para especificar el nombre de la acción que la actividad es capaz de realizar. Todos los *Intent Filter* deben tener un y sólo un nodo `action`. El nombre de una acción debería ser una cadena lo suficientemente descriptiva. Lo más correcto es utilizar un sistema de nombres basado en la nomenclatura de paquetes de Java.
- `category`: usa el atributo `android:name` para especificar bajo que circunstancias la actividad atenderá la solicitud. Cada elemento `intent-filter` puede contener múltiples elementos `category`. Los valores estándar de Android son los siguientes:
  - `ALTERNATIVE`: este valor indica que la actividad debería estar disponible como una alternativa de la acción por defecto realizada para el tipo de datos manejado. Por ejemplo, si la acción por defecto para un contacto es visualizarlo, nuestra actividad podría anunciar que es capaz de editarlo y presentarse como una alternativa para tratar ese tipo de dato. Esta alternativa se muestra normalmente en el menú de opciones de la actividad.
  - `SELECTED_ALTERNATIVE`: similar a la categoría anterior, pero para el caso de actividades que muestran un listado de elementos entre los que el usuario puede escoger, de tal forma que se pueda presentar una alternativa a la acción que se pueda realizar sobre el elemento seleccionado.
  - `BROWSABLE`: la acción sólo está disponible para un navegador web. Cuando un `Intent` se lance desde un navegador web siempre incluirá la categoría `BROWSABLE`.
  - `DEFAULT`: esto permite marcar una actividad como la acción por defecto para el tipo de datos especificado.
  - `GADGET`: esto indica que nuestra actividad puede ser ejecutada empotrada dentro de otra.
  - `HOME`: si especificamos esta categoría sin especificar una acción, estamos haciendo que la actividad represente una alternativa a la pantalla de inicio nativa.
  - `LAUNCHER`: usar esta categoría hará que nuestra actividad aparezca en el lanzador de aplicaciones de Android.
- `data`: este elemento permite especificar sobre qué tipos de datos puede actuar nuestra actividad. Se pueden incluir varios elementos de este tipo. Se puede utilizar cualquier combinación de los siguientes atributos:
  - `android:host` especifica un nombre de dominio válido.
  - `android:mimeType` permite especificar el tipo de datos que nuestra actividad puede manejar.

- `android:path` específica una ruta válida para la URI.
- `android:port` especifica puertos válidos para el host indicado.

1.4. Acceder al Intent dentro de una actividad

Cuando una actividad es iniciada por medio de un `Intent`, ésta necesita conocer qué acción realizar y sobre qué datos. Para ello es necesario acceder al `Intent` con el que se hizo la petición. Una forma de hacer esto es utilizar del método `getIntent` (lo cual se hará normalmente dentro del método `onCreate`):

```
@Override
public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);

        Intent intent = getIntent();
}
```

Para acceder a la acción y a los datos asociados al `Intent` haremos uso de los métodos `getData` y `getAction`. Los elementos extra pueden ser obtenidos por medio a llamadas a las funciones `get[TIPO]Extra`, donde `[TIPO]` es un determinado tipo de datos (String, Boolean, etc.):

```
String accion = intent.getAction();
Uri datos = intent.getData();
```

2. Navegación

Ya vimos en la sesión anterior que una aplicación suele estar compuesta de varias actividades, e introdujimos el concepto de *pila de actividades*. Dicha pila está compuesta por todas las actividades de nuestra aplicación que se han ido ejecutando. Cada vez que iniciamos una actividad ésta pasa a estar activa y en primer plano, haciendo que la que lo estuviera hasta ese momento pasara a ocupar el tope de la pila. Una vez que la actividad en primer plano termina, se destruye, y aquella que estuviera en el tope de la pila la abandona para pasar a ser la actividad activa.

Sin embargo, también es posible que una actividad inicie otra que forme parte de una aplicación distinta. Por ejemplo, si nuestra aplicación desea tomar una fotografía, se define un `Intent` para solicitar la realización de dicha acción. A continuación, una actividad de cualquier otra aplicación instalada en el dispositivo y que ha declarado previamente que se puede ocupar de este tipo de tareas se inicia, pasando a primer plano. Una ves que se toma la fotografía, nuestra actividad vuelve a primer plano y continua su ejecución. Para el usuario la sensación es como si la actividad encargada de tomar la fotografía formara parte de nuestra propia aplicación. A pesar de que las actividades en ejecución puedan pertenecer a diferentes aplicaciones, Android se encarga de navegar entre ellas de manera transparente al usiario manteniendo ambas actividades en una misma *tarea*.

Una **tarea** es una colección de actividades con las que el usuario interacciona para realizar una acción determinada. Cada tarea podría interpretarse como una aplicación independiente. Todas las actividades de una tarea se organizan en una pila de actividades. Toda tarea tiene, por lo tanto, su propia pila de actividades. Cuando el usuario selecciona el icono de una aplicación en el menú de aplicaciones, la tarea de dicha aplicación pasa a primer plano. Si no existiera una tarea para la aplicación, porque por ejemplo ésta no haya sido utilizada recientemente, se crea entonces una nueva tarea y la actividad principal de la aplicación pasa a estar asociada a la misma, activa y visible.

A partir de este momento las actividades de la aplicación se manejan mediante la pila de actividades tal como se explicó en la sesión anterior. Es realmente importante que recordemos cómo se realiza la navegación entre las actividades pertencientes a una tarea: cuando una actividad deja de estar activa pasa al tope de la pila; cuando la actividad activa termina, entonces la actividad en el tope pasa a estar activa.

Nota:

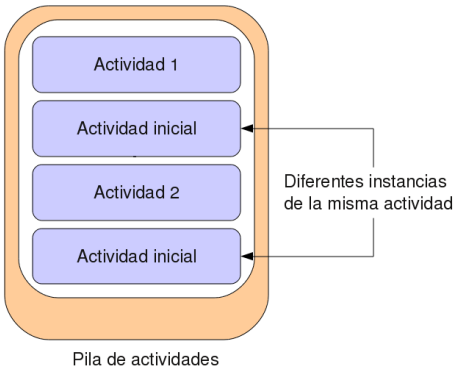
Lo anterior quiere decir que las actividades presentes en la pila nunca son reordenadas. Éstas son tan solo introducidas en el tope de la pila o extraídas del tope.

Una vez que todas las actividades son eliminadas de la pila de actvidades, la tarea correspondiente deja de existir. Una tarea se maneja como un bloque único que deja de estar en primer plano cuando el usuario inicia una nueva tarea o va a la pantalla de inicio de Android. Mientras la tarea no esté en primer plano, todas sus actividades están detenidas, pero en principio se mantienen intactas. De esta forma, cuando la tarea pasa a primer plano, el usuario puede seguir usándola como si nada hubiera pasado.

Nota:

Hemos de tener en cuenta que si el número de tareas que no se encuentra en primer plano crece, es posible que el sistema necesite eliminar algunas de sus actividades cuando se vea falto de recursos.

Debido a que nunca se reordenan las actividades en la pila de actividades, es posible que más de una instancia de una actividad se encuentre en ella. Esto puede suceder, por ejemplo, si más de una actividad de nuestra aplicación permite iniciar una actividad concreta. Cuando esto sucede, varias copias de la actividad se encuentran en la pila, por lo que cuando vamos pulsando el botón *BACK* de nuestro dispositivo cada instancia de dicha actividad se mostrará en el orden en el que fueron abiertas (cada una, por supuesto, con su propio estado de la interfaz). Este comportamiento está representado en la siguiente figura:



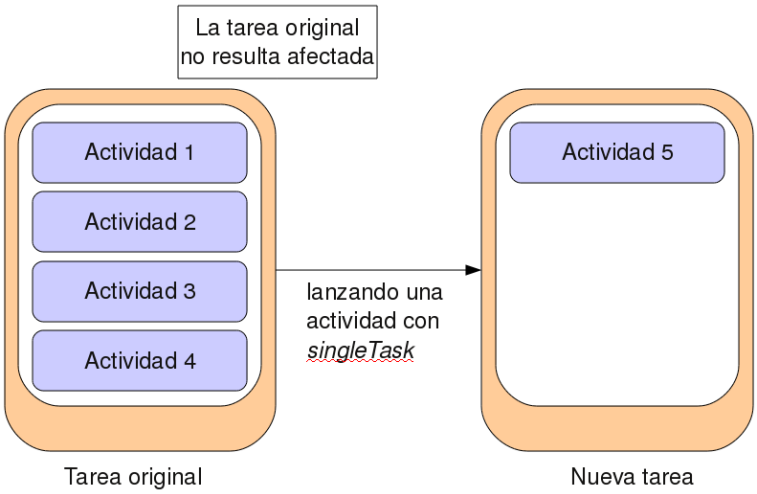
Ejemplo de varias instancias de una actividad en una pila

En esta sección veremos cómo modificar el comportamiento por defecto de Android a la hora de navegar entre actividades. Será posible, por ejemplo, hacer que una actividad sólo pueda ser instanciada una vez. O conseguir que una actividad se inicie en una nueva tarea en lugar de ser colocada en la tarea actualmente en ejecución. O incluso eliminar todas las actividades almacenadas en la pila de actividades cuando se lanza una determinada.

2.1. El atributo launchmode

Uno de los atributos del elemento `activity` en el *Manifest* de la aplicación es `launchmode`. Los diferentes valores de `launchmode` especifican cómo debería ser lanzada una actividad con respecto a su tarea. Los cuatro posibles valores de este atributo son:

- `standard`: se trata del modo por defecto. Cuando se lanza la actividad se crea una nueva instancia de la misma dentro de la tarea desde la que se inició y se le envía el `Intent`. La actividad puede ser instanciada múltiples veces; cada instancia puede estar en diferentes tareas, y también es posible disponer de varias copias de la actividad en una misma tarea.
- `singleTop`: si ya existía una instancia de la actividad en el tope de la pila de la tarea actual, se le envía el `Intent` a dicha instancia a través de una llamada a su método `onNewIntent`, en lugar de crear una nueva instancia. La actividad podría ser instanciada múltiples veces, tanto en diferentes como en la misma tarea; la única diferencia es que si la actividad ya se encontraba activa no se creará ninguna nueva copia de la misma. Esto evidentemente va a ser útil en el caso de actividades que puedan iniciarse a si mismas.
- `singleTask`: el sistema crea una nueva tarea y la nueva copia de la actividad pasa a ser la actividad activa en dicha tarea. Sin embargo, si ya existía una instancia de la actividad en una tarea separada, en lugar de crear una nueva copia se le envía a la ya existente el `Intent` por medio de una llamada a su método `onNewIntent`. Sólo puede existir una instancia de la actividad. Se podría interpretar el resultado como si se hubiera iniciado una nueva aplicación independiente de la anterior.



Ejemplo de singleTask

- `singleInstance`: el comportamiento es el mismo que en el caso de `singleTask`, con la diferencia de que el sistema nunca lanza ninguna otra actividad en la tarea que contiene la instancia recién creada. La actividad será siempre el único miembro de su tarea. Todas las actividades iniciadas por ésta se abrirán en una tarea nueva. Este valor se podría utilizar por ejemplo para una actividad que fuera a cumplir el papel de navegador web y que fuera a ser notificada con `Intents` desde diferentes tareas. Sólo existiría una copia de este navegador en la mamoria, que sería una aplicación totalmente independiente sin ninguna actividad adicional, y que en su historial incluiría todas las páginas que se hubieran visualizado a partir de diferentes aplicaciones.

Ejemplo de singleInstance

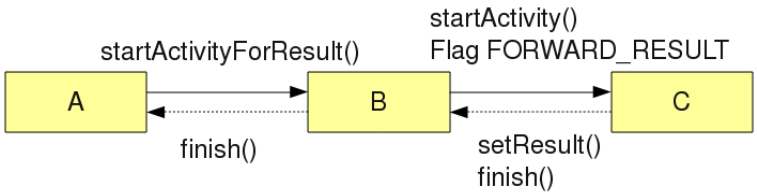
Nota:

Todos los comportamientos indicados anteriormente pueden ser sustituidos por otros cuando se añaden ciertos flags al `Intent` que lanza la actividad, los cuales se verán en la siguiente sección.

2.2. Modificar el lanzamiento de actividades mediante Intents

Además de mediante el método visto en la sección anterior, es posible modificar la forma en la que una actividad se asocia a una determinada tarea incluyendo un determinado flag en el `Intent` pasado como parámetro al método `startActivity`. Los flags que se pueden utilizar son los siguientes:

- `FLAG_ACTIVITY_NEW_TASK`: produce el mismo comportamiento que asignar el valor `singleTask` al atributo `launchMode`.
- `FLAG_ACTIVITY_SINGLE_TOP`: produce el mismo comportamiento que al asignar el valor `singleTop` al atributo `launchMode`.
- `FLAG_ACTIVITY_CLEAR_TOP`: si ya existe una copia de la actividad que se quiere ejecutar en la tarea actual, en lugar de lanzar una nueva copia se destruyen todas las actividades sobre ella en la pila de actvidades, y se le envía el `Intent` a dicha copia por medio de su método `onNewIntent()`.
- `FLAG_ACTIVITY_REORDER_TO_FRONT`: si existe una instancia de la actividad en la pila la llevará al primer plano haciéndola activa, sin necesidad de crear una nueva instancia de la misma.
- `FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS`: cuando se lanza una actividad en una nueva tarea, ésta normalmente se suele mostrar en la lista de actividades recientemente inciadas, a la cual se puede acceder mediante una pulsación larga del botón *HOME*. Usar este flag evitará que se produzca este comportamiento.
- `FLAG_ACTIVITY_FORWARD_RESULT`: ya sabemos que para que una subactividad pueda comunicarse con su actividad padre por medio del método `setResult` es necesario que ésta haya hecho uso del método `startActivityForResult`. Supongamos ahora el caso mostrado en la siguiente figura, en el que la actividad A inicia la actividad B, que a su vez inicia la actividad C. Si quisiéramos que el resultado devuelto de C a B por medio de `setResult` fuera también devuelto a A, deberíamos hacerlo manualmente volviendo a incluir una llamada a `setResult` en B. Para evitar tener que hacer esto podemos usar este flag. Para que esto suceda la actividad B debe haber sido lanzada con este flag y además **debe haber lanzado a la actividad C con `startActivity`** (y no `startActivityForResult`).



Ejemplo de FLAG_ACTIVITY_FORWARD_RESULT

- `FLAG_ACTIVITY_MULTIPLE_TASK`: este flag no tiene efecto a menos que se utilice conjuntamente con `FLAG_ACTIVITY_NEW_TASK`. Con este flag siempre se crea una nueva instancia de la actividad que será colocada en el tope de la pila de una nueva tarea.

- `FLAG_ACTIVITY_NO_ANIMATION`: este flag desactiva la animación que normalmente se suele mostrar cuando se pasa de una actividad a otra. Esto podría ser utilizado por ejemplo en el caso de lanzar una actividad que a su vez va a lanzar otra sin ninguna interacción por parte del usuario. El resultado sería que se produciría una única animación de transición entre actividades en lugar de dos.

2.3. Afinididad entre actividades

La afinidad es un medio para indicar a qué tarea prefiere pertenecer una actividad. Todas las actividades de una misma aplicación tienen por defecto afinidad con el resto. Así que por defecto todas las actividades de una misma aplicación prefieren pertenecer a la misma tarea. Sin embargo, este comportamiento se puede modificar, de tal forma que actividades definidas en diferentes aplicaciones puedan compartir afinidad o que actividades definidas en una misma aplicación puedan tener diferentes afinidades y por lo tanto asignarse a diferentes tareas. Para ello se debe hacer uso del atributo `taskAffinity` del elemento `activity` en el *Manifest* de la aplicación.

El valor del atributo `taskAffinity` debe ser una cadena, que debe ser diferente del nombre del paquete por defecto definido en el elemento `manifest` dentro del *Manifest* de la aplicación. Esto es así porque ese nombre es el que la aplicación que utiliza por defecto para definir la afinidad de sus tareas.

La afinidad entre actividades entra en juego cuando se presenta cualquiera de las siguientes circunstancias:

- Cuando el `Intent` que inicia una nueva actividad contiene el flag `FLAG_ACTIVITY_NEW_TASK`. Cuando esto sucede normalmente se crea una nueva tarea para la actividad, pero esto no tiene por qué ser así. Si ya existía una tarea en la que sus actividades comparten la misma afinidad que la nueva actividad, ésta se inicia en dicha tarea. En otro caso se crea una nueva tarea. Hemos de tener cuidado con esto. Si al usar este flag se crea una nueva tarea para nuestra actividad, deberemos proporcionar algún medio para poder volver a mostrar la actividad en el caso en el que el usuario vuelva a la pantalla inicial de Android por medio del botón correspondiente de su dispositivo. Una forma de hacerlo sería por ejemplo incluir un icono en alguna parte para poder volver a ella.
- Cuando el atributo `allowTaskReparenting` de la actividad tiene el valor `"true"`. En este caso la actividad podría moverse de la tarea que inicia a la tarea por la que tenga afinidad, cuando esta tarea pase a primer plano.

Nota:

Si un paquete `.apk` contiene más de una *aplicación* desde el punto de vista del usuario, quizá sea interesante hacer uso de `taskAffinity` para asignar diferentes tareas a las actividades correspondientes a cada *aplicación*

2.4. Limpiar la pila de actividades

Si una tarea deja de estar en primer plano durante un periodo prolongado de tiempo, el sistema eliminará todas las actividades de la tarea excepto la actividad que deba pasar a activa cuando la tarea vuelva al primer plano. Por lo tanto, cuando se vuelva a activar la tarea, tan sólo se restaura dicha actividad. Esto es así porque el sistema supone que tras estar mucho tiempo sin utilizar una tarea probablemente se haya abandonado para hacer cualquier otra cosa. Sin embargo, existen ciertos atributos del elemento `activity` en el *Manifest* de la aplicación que permiten modificar este comportamiento:

- `alwaysRetainTaskState`: si se asigna el valor `true` a este atributo para la actividad en el tope de la pila de la tareas, el proceso descrito anteriormente nunca tiene lugar. La tarea mantiene todas sus actividades incluso tras un largo periodo de tiempo.
- `clearTaskOnLaunch`: si se asigna el valor `true` a este atributo para la actividad en el tope de la pila de tareas, se produce el efecto contrario que en el caso del atributo anterior; tan pronto como la tarea deje de estar en primer plano se eliminarán todas las actividades de la pila excepto aquella en el tope de la misma. Cada vez que se abandona la tarea y se vuelve a ella ésta se encontrará en su estado inicial, incluso si ha sido sólo durante un corto periodo de tiempo.
- `finishOnTaskLaunch`: este atributo cumple una función parecida a la de `clearTaskOnLaunch`, pero operando a nivel de una sóla actividad y no de toda la tarea. Si su valor es `true`, tan pronto la tarea deje de estar en primer plano se destruirá la actividad.
- `noHistory`: este atributo también se aplica a una actividad individual, y es muy parecido al anterior, pero no igual. En este caso, si su valor es `true` para una determinada actividad, la actividad se destruirá *cuando deje de estar en primer plano* (la actividad, no su tarea). Esto quiere decir que la actividad nunca llegará a estar en la pila de tareas.

3. Esquemas típicos de navegación

Una vez estudiados todos los conceptos anteriores, describimos brevemente en esta sección algunos esquemas básicos de navegación en Android, viendo que sucede cuando pulsamos los botones *HOME*, *BACK* o cambiamos de una aplicación a otra. Esto nos permitirá comprender mejor cómo debemos plantear la navegación en nuestras aplicaciones y que consideraciones debemos tomar a la hora de diseñarlas.

3.1. Iniciando una aplicación desde la pantalla inicial de Android

La pantalla inicial de Android es el punto a partir del cual podremos lanzar muchas de nuestras aplicaciones (algunas aplicaciones sólo pueden ser lanzadas desde otras). Cuando el usuario pulsa sobre un icono en el lanzador de aplicaciones (o sobre su correspondiente icono en la pantalla inicial), se lanza la actividad principal de dicha

aplicación; ésta se pone en primer plano y puede captar la entrada del usuario. La actividad correspondiente a la pantalla de inicio permanecerá en segundo plano, detenida, a la espera de que el usuario pulse la tecla *HOME* en su dispositivo.

3.2. Abandonar una actividad con los botones BACK y HOME

El efecto que abandonar una actividad tendrá sobre ésta dependerá de cómo lo haga el usuario. Por ejemplo, pulsar el botón *BACK* del dispositivo hará que por defecto la actividad en primer plano se destruya y se muestre la anterior en la pila de actividades correspondiente. En el caso en el que la pila quede vacía, se elimina la tarea asociada y se muestra la primera de las actividades de la siguiente tarea. En el caso en el que no hubiera ninguna otra tarea en ejecución simplemente se vuelve a mostrar la pantalla de inicio de Android. Esto quiere decir que si cerramos una aplicación por medio del botón *BACK*, ésta volverá a ser mostrada en su estado inicial si se vuelve a lanzar, porque fue destruida.

Sin embargo, si se pulsa *HOME* en lugar de *BACK*, se vuelve a la pantalla incial, pasando la actividad que estuviera activa a segundo plano pero **sin destruirla**. El efecto que producirá ejecutar de nuevo la aplicación será simplemente volver a llevar a primer plano la tarea correspondiente, con lo que la actividad correspondiente se volvería a mostrar en el estado en el que se encontrara.

Esto permite la posibilidad de ejecución **multitarea** en Android. Esto se produce si al volver a la pantalla inicial de Android por medio del botón *HOME* se decide lanzar una aplicación distinta a la que ha quedado fuera del primer plano. Cada aplicación sería lanzada en una tarea diferente. Ahora seríamos perfectamente capaces de pasar de una aplicación a la otra, pasando las tareas correspondientes de primer a segundo plano y viceversa.

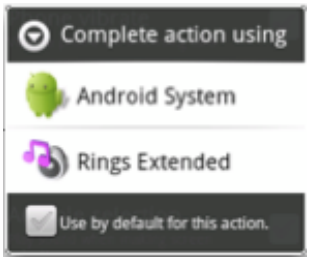
Existen excepciones para este comportamiento. Algunas actividades se vuelven a mostrar en su estado inicial tras volverlas a colocar en primer plano tras haber dejado de estar activas previamente. Esto puede pasar por ejemplo con los contactos. Si se selecciona un contacto para ver sus detalles y se cierra la aplicación de contactos, al volver a iniciarla se mostrará de nuevo la lista de contactos, y no los detalles del contacto que se estuviera consultando en el momento de cerrar la aplicación. Por otra parte, no todas las actividades tienen que ser destruidas tras pulsar el botón *BACK*. Por ejemplo, en el caso de un reproductor de música, éste podría seguir en funcionamiento en segundo plano mientras la música sigue sonando aunque se hubiera pulsado el botón *BACK*.

3.3. Reutilizar y reemplazar actividades

Cuando una actividad A lanza una actividad B perteneciente a una aplicación distinta, diremos que A reutiliza B. Esto sucede como ya hemos visto cuando A no es capaz de realizar una determinada tarea y confía en B para ello. A la hora de diseñar una actividad es importante dedicar un tiempo a pensar cómo podría reutilizar otras o cómo funcionaría

a la hora de ser reutilizada por actividades de terceros.

El caso contrario al anterior se conoce como reemplazar una actividad. Esto sucede cuando una actividad A está más capacitada para llevar a cabo una tarea que otra actividad B. En este caso A y B tienen un nivel de equivalencia tal que A podría ser utilizada en lugar de B. Esto contrasta con el caso anterior en el que A y B son actividades totalmente diferentes que se complementan. Por ejemplo, supongamos que el usuario se ha descargado una aplicación que puede sustituir a la aplicación nativa para los tonos de teléfono. Cuando el usuario intenta acceder a la configuración de los tonos de teléfono a través de las opciones de configuración, verá un cuadro de diálogo que le permitirá escoger entre esta nueva aplicación o la nativa de Android, con una opción para recordar la opción escogida en el futuro.



Reemplazo de actividades