Intents

Componentes

Antes de meternos con los *intents*, tenemos que comprender lo que es un **componente**. La versión abreviada es que el propio S.O. de Android y todas las apps que pueda tener instaladas un dispositivo concreto son o tienen componentes. Android permite, siguiendo unas reglas, que un componente llame a otro componente. Por ejemplo, si quieres que tu App utilice la cámara de fotos del móvil, no tienes que escribir el código o usar una librería. Puedes 'preguntar' a ver si otras apps son capaces de hacer el trabajo por ti. Esto es lo que se llama **comunicación entre componentes**, y sirve entre otras cosas, para que tu app pueda funcionar con cualquiera de las cámaras de fotos de cualquier dispositivo móvil del mercado.

La definición de libro de componente es la siguiente: los **componentes de aplicación** son los bloques de construcción esenciales de una app. Estos *componentes* están relacionados de forma débil a través del fichero AndroidManifest.xml. Los componentes más importantes que pueden usarse dentro de una app Android son:

- **Actividades**: Hasta ahora hemos vistos únicamente apps con una sola *actividad*, pero pueden tener varias. Las *actividades* pueden ser ejecutadas por otras *actividades* si se siguen los pasos adecuados.

- **Servicios**: Los *servicios* son apps que se ejecutan sin interfaz, en background. Podríamos decidir que el código para conectarnos a una wifi para comprobar continuamente el precio de unas acciones sea un *servicio*.

- **Proveedores de Contenido**: Manipulan una serie de datos que pueden ser compartidos por nuestra App. Por ejemplo, un *proveedor de contenidos* puede ser la lista de contactos de nuestro móvil. Nuestra App podría usarla para crear nuevos contactos, leerlos, etc.

- **Receptores de Broadcast:** Apps que reciben mensajes de multidifusión de otras apps. Podrían ser por ejemplo apps que avisan de que te ha llegado tu pedido de Amazon, o que tienes un mail, etc.

Nosotros al menos al principio sólo vamos a trabajar con *actividades*, el resto de componentes los veremos más adelante. Cuando una *actividad* intenta mandar un mensaje a otra *actividad*, a eso se le llama **intent**. La comunicación a base entre intents es **asíncrona**, lo que puede dar lugar a problemas de comunicación, sincronización, etc. Los *intents* también pueden usarse para comunicar una *actividad* con un *servicio*.

Intents

La clase **Intent** describe una operación que queremos hacer como iniciar una actividad o servicio, intercambiar datos entre apps o componentes, o solicitar al S.O. que haga determinada operación.

Los posibles usos de un *intent* se clasifican así:

- Si deseas arrancar un componente concreto es posible nombrarlo de forma **explícita**. Se usan típicamente para arrancar *actividades* de nuestra propia app. Por ejemplo, en una app con dos *actividades* podríamos ir de una a otra imitando lo que sería un "cambiar de ventana".

- En lugar de nombrar al componente que se quiere arrancar, se puede declarar la intención de comenzar una acción genérica de forma **implícita**. Gracias a los *intents implícitos* se hace posible la **reusabilidad**. Es similar a preguntar al Sistema Operativo a ver quién es capaz de usar la cámara de fotos, y que éste nos diga qué otros componentes (incluso desde otras apps) tiene registrado que son capaces de hacerlo.

- Anunciar al resto de sistema operativo que se ha producido determinado **evento** de forma que las *actividades* puedan reaccionar a él. Por ejemplo, cuando el nivel de batería es bajo o se pierde la conexión a Internet.

Intents Explícitos [Java] – Sin respuesta

Un **Intent Explícito** sucede cuando indicamos qué componente queremos invocar; normalmente otra *actividad* de la misma app. Cuando tengamos que delegar una tarea de una *actividad* a otra *actividad* recurrimos al **intent.**

En nuestro proyecto, para agregar otra *actividad* vamos al explorador de proyectos, botón derecho, **New-> Activity-> EmptyActivity**. Esto crea una clase Java, un layout y además **registra** esa nueva *actividad* en el **AndroidManifest.xml**. Nunca hagas una actividad nueva a mano para evitar problemas.

Vamos a construir un ejemplo en el que una *actividad* realiza un intent sobre otra *actividad*. Creamos un botón con un evento que lanza el intent. Necesitaremos pasarle el contexto, y la actividad a la que queremos llamar. Vamos a pasarle además en el mensaje un valor adicional etiquetado como *Veces*. Podemos pasar a la *activity* destino virtualmente lo que queramos. Finalmente, usamos el método **startActivity ()** para iniciar la nueva *actividad*.

Como ves, es muy sencillo. El efecto visual es que la app cambia de una pantalla a otra.

```
buttonToAc2.setOnClickListener( new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(getApplicationContext(), Activity2.class);
        intent.putExtra ( name: "Veces", finalVeces );
        startActivity(intent);
    }
});
```

No, es mentira. Es más complicado de lo que parece. Lo que realmente ocurre en la app es lo siguiente:

1. Se inicializa la nueva *actividad*, ejecutándose su **onCreate ()**.
2. La *actividad* original se va a **la pila**.
3. Si en algún momento la nueva *actividad* hace **finish ()**, o el usuario pulsa al botón de atrás del móvil; entonces la nueva *actividad* se cierra y se saca de la pila.
4. Se vuelve a mostrar la *actividad* original.

Date cuenta de que cada vez que hacer un *intent* estás inicializando una nueva actividad. Es posible que quieras 'ir y volver' a la misma *actividad*, lo cual es correcto, pero **es un error de libro** que solamente pongas un intent para ir de una *activity1* a otra *activity2*, y otro *intent* para ir de la *activity2 a la activity1*. De esta forma no estás volviendo a la *activity* anterior: estarías creando nuevas actividades cada vez, llenando la pila y ocupando memoria.

La forma correcta de 'volver' o 'reutilizar' a la actividad anterior sería usando **finish ()**, que finaliza la actividad en curso.

```
buttonEnd.setOnClickListener( new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        finish();
    }
});
```

Por supuesto, si tienes claro lo que estás haciendo, nada te impide finalizar la actividad en curso en cualquier momento si no tienes pensado 'volver' a ella en ningún momento. Esto puede hacerse y es correcto.

```
buttonToAc2.setOnClickListener( new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(getApplicationContext(), Activity2.class);
        intent.putExtra ( name: "Veces", finalVeces );
        startActivity(intent);
        finish();
    }
});
```

Por último, ten en cuenta que **finish ()** sobre una *actividad* ejecuta automáticamente su método **onDestroy ()**. Aprovecha esto para salvar el estado de la *actividad*, por ejemplo.

Los valores pasados en el mensaje de una *actividad* a otra pueden ser recuperados del *intent*. Nótese que en caso de que no existan valores pasados o la etiqueta sea errónea, los valores estarán a nulo.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate( savedInstanceState );
    setContentView( R.layout.activity_2 );

    Bundle extras = getIntent().getExtras();
    String veces = extras.getString( key: "Veces");
```

Intents Explícitos [Java] – Con respuesta

Una *actividad* iniciada mediante **startActivity ()** es independiente de la *actividad* que la lanzó originalmente, por lo tanto, no la proporcionará ningún tipo de información cuando finalice. Existe no obstante una forma de vincularlas, de forma que cuando la nueva *actividad* finalice, lance un **evento** que será recogido por la *actividad* original. Obviamente, podrá enviar resultados a dicha *actividad* a modo de respuesta.

En lugar de **startActivity ()**, usaremos el método **startActivityForResult ()**, que es similar salvo porque se le añade un **código de petición** para saber más tarde qué *subactividad* ha finalizado (una *actividad* puede inicializar varias *subactividades*, al fin y al cabo).

En el ejemplo anterior, podríamos hacer lo siguiente. Primero, definimos el código:

```
public static final int SUBACTIVITY_1 = 1;
```

Y después, simplemente realizamos el *intent*.

```
buttonToAc2.setOnClickListener( new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(getApplicationContext(), Activity2.class);
        intent.putExtra ( name: "Veces", finalVeces );
        startActivityForResult(intent, SUBACTIVITY_1);
    }
});
```

Nota: el método está deprecado, pero nos da igual. Estamos aprendiendo.

Cuando queramos terminar la *actividad*, podremos indicar el estado en el que la *actividad* ha finalizado (correctamente, cancelada, etc.) y pasar parámetros a la *actividad* principal mediante *intents*, tal y como se ve a continuación.

```
buttonOk.setOnClickListener( new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(getApplicationContext(), MainActivity.class);
        intent.putExtra ( name: "Veces", veces);
        setResult(RESULT_OK, intent);
        finish();
    }
});

buttonCancel.setOnClickListener( new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        setResult( RESULT_CANCELED );
        finish();
    }
});
```

¿Recuerdas que dijimos que **startActivityForResult ()** genera un **evento** cuando la *actividad* llamada finaliza? Dicho evento se define de la siguiente forma:

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

    if (requestCode == SUBACTIVITY_1){

        if (resultCode == RESULT_OK){
            // Todo ha ido bien
            String veces = data.getStringExtra( name: "Veces" );

        } else if (resultCode == RESULT_CANCELED){
            // Tratamos el error
        }
    }
}
```

Preguntando por cada uno de los valores podemos distinguir qué subactividad ha finalizado, en qué estado ha quedado, y por supuesto, recuperar los valores retornados a través del intent (data).

Práctica 28

[Java] Realiza la práctica **07 – Intents Explícitos [Java]**.

Práctica 29

[Java] Realiza la práctica **08 – Intents Explícitos II [Java]**.