Intents e Intent Filters

Un Intent, es una descripción de una acción que se quiere realizar, por sí solo, éste no hace nada, sino que describe algo que tiene que ocurrir. Esta entidad se representa mediante un objeto.

Un IntentFilter, es la definición de una acción que una aplicación es capaz de llevar a cabo. Los IntentFilters de una aplicación, declaran qué tipo de acciones puede realizar dicha aplicación. Estos filtros se declaran como información de la aplicación en el Manifest.

Funcionamiento

Cuando una aplicación quiere realizar una acción, ésta declara su "intención" de realizarla mediante un Intent, y lo envía al OS. El sistema, al recibir este Intent, busca qué otra u otras aplicaciones pueden llevar a cabo dicha acción, analizando los IntentFilters de todas las aplicaciones disponibles.

En el caso de existir más de una aplicación que pueda llevar a cabo la tarea, el sistema le preguntará al usuario con cuál de todas ellas desea realizar la acción.

Tipos de Intents

- **Explícitos:** Se declara el nombre de una clase en particular que se quiere ejecutar.
- **Implícitos:** Se declara una acción a ejecutarse, sin saber qué aplicación la llevará a cabo.

Intents explícitos, ejecutando Activities

La forma de disparar una Activity desde otra Activity es mediante un Intent, en el cual se pasará como parámetro el nombre de la clase que se quiere ejecutar.

Dentro de la primera Activity:

```
Intent i = new Intent(this,Pantalla2Activity.class);
startActivity(i);
```

El primer parámetro del constructor del Intent es el Contexto, en este caso se pasa la Activity, y el segundo parámetro, es el objeto Class de la clase que queremos ejecutar*, el cual lo obtenemos agregando ".class" el nombre de la clase.

Una vez creado el objeto Intent, se ejecuta el método "startActivity()", perteneciente a la clase Activity, el cual recibe como parámetro el Intent anterior. En ese momento se disparará la segunda Activity.

* *Todo objeto puede devolver un objeto del tipo Class mediante la sintaxis ".class" después del nombre del tipo de dato o ejecutando el método ".getClass()" a un objeto, este objeto Class se utiliza en lo que en Java se conoce como Reflection.*

NOTA: No debe olvidarse declarar la segunda Activity en el archivo manifest.xml, ya que de lo contrario, al disparar el Intent, se lanzará una excepción y la aplicación terminará.

```
<activity
    android:label="@string/app_name"
    android:name=".Pantalla2Activity" />
```

Agregamos la segunda Activity en el manifest, en este caso, la clase es llamada "Pantalla2Activity"

Pasando información entre Activities

Si la información es de tipos de datos primitivos, se pasa directamente por medio de putExtra():

```
Intent i = new Intent(this,Pantalla2Activity.class);
i.putExtra("clave1","valor String");
i.putExtra("clave2",53);
startActivity(i);
```

El cual recibe 2 parámetros, "clave" y "valor", la clave siempre es un String, y el valor, puede ser cualquier tipo de dato primitivo.

¿Cómo llegan estos datos a la Activity que se dispara? Esta Activity podrá recuperar el objeto Intent mediante el método "getIntent()"

Código en la segunda Activity:

```
Intent intent = getIntent();
```

Luego se pide al Intent los extras, los cuales están contenidos en un objeto tipo "Bundle"

```
Bundle extras = intent.getExtras();
```

Este objeto Bundle, es el que contiene la información pasada a la Activity, y la cual puede obtenerse mediante el nombre de las claves:

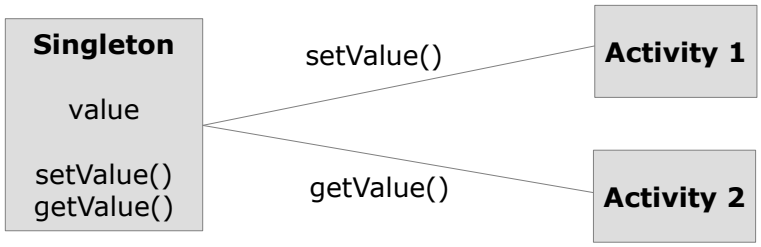
```
String strActivity1 = extras.getString("clave1");
int intActivity1 = extras.getInt("clave2");
```

El problema surge cuando se quieren pasar tipos de datos que no son los primitivos, existen varias alternativas para pasar información entre Activities:

- Una clase Singleton.
- Un Atributo estático en una clase.
- HashMap.
- Application Object

Una clase Singleton

Como todos los componentes de la aplicación corren en el mismo proceso, se puede tomar ventaja de ésto al utilizar un Singleton. Recordemos que un Singleton es una clase que posee solo una instancia. Esto permite a todas las Activities obtener la misma instancia del Singleton, y utilizarlo como punto de acceso compartido a la información.
Por ejemplo, una Activity puede pasar un valor al Singleton mediante un método setValue(valor) y una segunda Activity puede recuperar ese valor mediante un método getValue() del mismo singleton.



Un campo o método público y estático

Puede definirse el valor que se quiere compartir como un atributo público y estático de la primer Activity, de modo que la segunda pueda acceder a dicho atributo:

Un HashMap

Puede definirse una clase que contenga un objeto HashMap público y estático (para que pueda ser accedido desde el resto de las clases). La primera Activity pondrá un valor en el HashMap y pasará la clave a la segunda Activity mediante putExtra().

Application Object

Toda aplicación puede tener instanciada una clase que herede de Application. Solo basta declarar en el archivo manifest.xml el nombre de nuestro clase, y el sistema se encargará de instanciarla en el momento de la ejecución de la aplicación.

Definimos la clase "MyApp":

```
public class MyApp extends Application
{
      //...
}
```

Definimos la clase en el manifest:

```
    <application
        android:icon="@drawable/ic_launcher"
        android:name=".MyApp"
        android:label="@string/app_name" >
```

Luego podremos definir dentro de la clase MyApp todos los atributos que queramos y que serán usados para pasar información entre Activities.

```
public class MyApp extends Application
{
        private String value;

        public String getValue() {
              return value;
       }

       public void setValue(String value) {
              this.value = value;
       }

       //...
```

También definimos el constructor de la clase, y el hacemos override del método "onCreate()", en los cuales debemos llamar a los métodos de la clase padre mediante "super":

```
        public MyApp()
        {
              super();
        }

        @Override
        public void onCreate()
        {
              super.onCreate();
        }
}
```

Antes de disparar la segunda Activity, obtenemos la instancia de la clase Application y utilizamos el método "setValue()" para dejar guardada la información que queremos que la segunda Activity recupere:

```
Intent i = new Intent(this,Pantalla2Activity.class);
MyApp app = (MyApp) getApplication();
app.setValue("String desde Activity 1");
startActivity(i);
```

En cualquier Activity, podremos obtener la instancia de la clase MyApp llamando al método "getApplication()" el cual devuelve un objeto del tipo Application, por lo que habrá que castear el objeto a "MyApp"

En la segunda Activity, se recupera el objeto Application de la misma forma y se obtiene el valor mediante el getter:

```
String value = app.getValue();
```

Obteniendo datos desde la segunda Activity

Muchas veces se lanza una segunda Activity para obtener información que ingresa el usuario (por ejemplo), y al cerrarse esta Activity, esta información debe ser devuelta a la Activity que la disparó, para realizar esto, existen dos formas posibles:

- Se utiliza alguno de los métodos anteriormente descriptos (Singleton, Application object, etc.).
- Se dispara la Activity mediante el método "*startActivityForResult*()" en vez de utilizar "StartActivity()".

Explicaremos la segunda opción:

El método "*startActivityForResult*()" tiene dos parámetros: el Intent, y un "requestCode" el cual es un int que el programador puede utilizar para identificar el resultado cuando llegue.

El resultado llegará como un evento a la primera Activity, por lo que ésta deberá hacer Override del método "onActivityResult()".

Disparando la segunda activity:

```
Intent i = new Intent(this,Pantalla2Activity.class);
startActivityForResult(i, 0);
```

En este caso se utiliza un requestCode igual a cero.

En la segunda Activity, debemos crear un intent, donde colocaremos la información a devolver mediante putExtra():

```
Intent intent = new Intent();
intent.putExtra("clave","texto");
```

Luego, pasamos este Intent con la información al método perteneciente a la Activity "setResult()", el cual también tiene como parámetro una constante que puede tomar los siguientes valores:

- RESULT_OK
- RESULT_CANCELED

```
this.setResult(RESULT_OK, intent);
finish();
```

Por último, se llama al método "finish()", terminando así la ejecución de la segunda Activity y se devolverá a la ejecución de la primera. Al ocurrir esto, se llamará al método "onActivityResult()" en donde la primer Activity podrá obtener el resultado de la segunda.

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

	if(requestCode==0 && resultCode==RESULT_OK)
	{
		String texto = data.getExtras().getString("clave");
		//...
	}
}
```

Podemos observar que este método nos proporciona 3 variables, "requestCode" será el valor pasado al método "startActivityForResult()" cuando se disparó la Activity, "resultCode" es el valor seteado en el método "setResult()" en la segunda Activity, y por último, el objeto Intent que creo la segunda Activity.

NOTA: También pueden definirse constantes RESULT personalizadas. Para ello existe la constante llamada "RESULT_FIRST_USER", que es un número a partir del cual pueden definirse los RESULTS personalizados.

Intents implícitos, ejecutando Aplicaciones

Al disparar un Intent indicando qué acción se quiere realizar y dejando que el S.O. se encargue de encontrar la aplicación que realice la tarea, existen dos parámetros importantes que el Intent debe recibir:

- action
- data

"action", es la acción que quiere realizarse, existen constantes definidas para acciones definidas, por ejemplo:

- ACTION_VIEW : Muestra información.
- ACTION_DIAL : Muestra el marcador de números telefónicos.
- ACTION_SEND : Envía información.

"data" es información que necesita la aplicación que va a realizar la tarea (por ejemplo, para marcar un número de teléfono, es necesario tener dicho número). Es una URI.

Ademas de estos dos parámetros principales, existen mas atributos que se pueden incluir en el Intent:

- category
- type
- component
- extras

"category" es información adicional para la acción que se va a ejecutar. Indica qué tipo de aplicación se esta buscando para realizar la acción. Algunos valores posibles son:

- CATEGORY_LAUNCHER : La aplicación tiene una Activity que se ejecuta al iniciar.
- CATEGORY_PREFERENCE : La aplicación es un panel de opciones.
- CATEGORY_HOME : La aplicación muestra el Home del OS.

"type" especifica un tipo de dato para el atributo "data", muchas veces, la información puesta en "data" es suficiente para que el sistema sepa de qué tipo de datos de esta hablando y no es necesario este parámetro. Se aclara como el Content-type de un header MIME*, por ejemplo:

- text/*
- image/png
- text/html

"component" es donde colocaremos explícitamente el package de una clase (que es el nombre de una aplicación) para que utilice el Intent. Normalmente, este valor es deducido por el S.O. a partir de los demás ítems del Intent.

"extras" es información extra mediante un objeto Bundle, es decir que la información se carga mediante clave-valor. Por ejemplo, si se desea enviar un e-mail, en los extras deberá indicarse dirección,asunto,texto,etc.

*Multipurpose Internet Mail Extensions o MIME son una serie de convenciones o especificaciones dirigidas al intercambio a través de Internet de todo tipo de archivos (texto, audio, vídeo, etc.) de forma transparente para el usuario. El encabezado Content-type indica el tipo de medio que representa el contenido del mensaje, consiste en un tipo: type y un subtipo: subtype, por ejemplo:

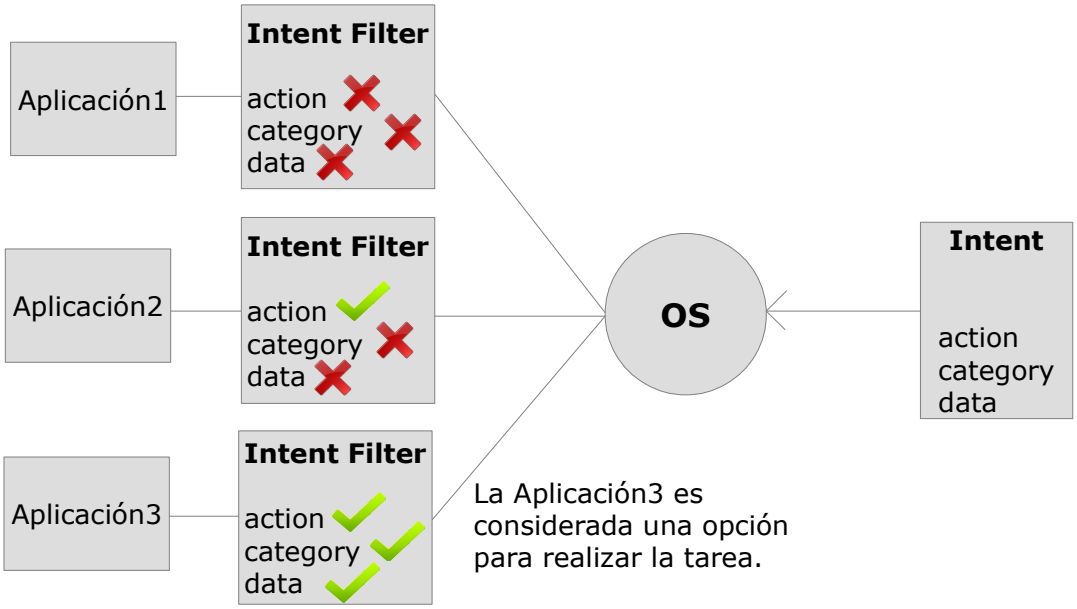
Content-Type: text/plain

Intent Filters

En los Intents implícitos, el S.O. debe encontrar cuál o cuales son las aplicaciones que pueden realizar la tarea que se describió en el Intent. Para ello, compara los 3 parámetros descriptos en el Intent (action, data y category) con los Intent Filters de cada aplicación, éstos describen mediante los mismos parámetros, qué tareas puede realizar dicha aplicación, si existe una coincidencia, entonces el S.O. la elige como una opción para realizar la acción.

Cada Intent Filter describe la capacidad de un componente de hacer algo.

Los Intent Filters se declaran en el archivo Manifest.xml, como se mencionó anteriormente. Éstos poseen 3 campos a ser definidos: action, data y category. A partir de éstos el S.O. considera a la aplicación como una opción para realizar una tarea. **Los 3 campos deben coincidir** con los del Intent lanzado.



Definición en el manifest

Por defecto, al crear un proyecto con Eclipse, se crea una Activity, la cual posee en el Manifest un Intent Filter:

```
<activity
    android:label="@string/app_name"
    android:name=".Pantalla1Activity" >
    <intent-filter >
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Cada Activity puede tener muchos Intent Filters. En este caso, el filtro indica al S.O. que esta Activity es la primera que se ejecuta al disparar la aplicación (MAIN) y que se debe poner en la lista de aplicaciones del S.O. (LAUNCHER)

Atributo "data":
Es el que contiene una URI*, especificada mediante "android:scheme". También se aclara el MIME type mediante "android:mimeType"

```
<intent-filter>
      <data android:mimeType="video/mpeg" android:scheme="http" . . . />
      <data android:mimeType="audio/mpeg" android:scheme="http" . . . />
      ...
</intent-filter>
```

El OS compara tanto la URI como el Type:

Intent	***Intent Filter***	***Pasa***
Sin data	**Sin data**	**SI**
Solo URI	Sin data	NO
Solo Type	Sin data	NO
URI y Type	Sin data	NO
Sin data	Solo URI	NO
Solo URI	**Solo URI**	**SI**
Solo Type	Solo URI	NO
URI y Type	Solo URI	NO
Sin data	Solo Type	NO
Solo URI	Solo Type	NO
Solo Type	**Solo Type**	**SI**
URI y Type	Solo Type	NO
Sin data	URI y Type	NO
Solo URI	URI y Type	NO
Solo Type	URI y Type	NO
URI y Type	**URI y Type**	**SI**

En la tabla mostramos a la izquierda el intent generado por el programador, y mostramos todas las combinaciones de las 2 informaciones que este intent puede tener (uri y type), y a la derecha, todas las combinaciones que pueden definirse en el Intent Filter de alguna otra aplicación, en la tercer columna, se indica si con las combinaciones de información del Intent y del Filter, el Sistema operativo acepta como opción del intent generado, a la aplicación que posee el Intent Filter con la información indicada para llevar a cabo la tarea.

Algunos ejemplos:

```
<data android:mimeType="image/*" />
```

Le dice al OS que la aplicación trabaja con imágenes de cualquier tipo.

```
<data android:scheme="http" android:type="video/*" />
```

Le dice al OS que la aplicación trabaja con videos desde internet.

*Uniform Resource Identifier o URI es una cadena de caracteres corta que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.).

Como crear Intents implícitos desde una Activity

Existen dos formas de disparar un Intent implícito para que se ejecute una aplicación:

- Disparando el Intent y dejando que el S.O. busque cuales aplicaciones tienen la capacidad de resolver la tarea, y que pregunte al usuario cual de todas quiere ejecutar si se encuentra más de una.
- Disparando el Intent y obligando al S.O. que muestre el cuadro de diálogo para que el usuario elija la aplicación. (Esto es debido a que puede ser que exista una aplicación que fue marcada como "por defecto" para realizar la acción, y aunque haya más de una aplicación disponible, el S.O. directamente ejecuta la marcada por defecto.)

Con la primera opción:

Explicaremos un ejemplo de como disparar un Intent para enviar un mensaje de texto.

Primero creamos un Intent pasándole la acción:

```
Intent i = new Intent(Intent.ACTION_SEND);
```

Luego seteamos el data type mediante el método "setType()":

```
i.setType("text/*");
```

Por último, incluimos información extra que necesita la aplicación que enviará el mensaje.

```
i.putExtra(Intent.EXTRA_SUBJECT, "asunto");
i.putExtra(Intent.EXTRA_TEXT, "texto del mensaje");
```

Y disparamos el Intent, para que el OS se encargue:

```
startActivity(i);
```

Con la segunda opción:

Para obligar al sistema a que muestre el diálogo para elegir una aplicación al usuario, utilizaremos el método estático "createChooser()" perteneciente a la clase Intent. En esta oportunidad mostraremos un ejemplo de como abrir una pagina web:

```
Intent i = new Intent(Intent.ACTION_VIEW);

i.setData(Uri.parse("http://www.google.com"));

startActivity(Intent.createChooser(i, "Elija un navegador"));
```

En este caso, seteamos la acción en VIEW, y pasamos como data, la URI correspondiente a la dirección de una pagina web. El método "createChooser()" recibe el Intent creado y devuelve otro Intent, el cual será el que dispararemos. En este caso, si el OS encuentra más de un navegador, aparecerá el diálogo para elegir uno.

NOTA: El emulador no muestra el diálogo.

Intents para compartir (Share)

Existe una clase llamada *ShareCompat*, que nos permite crear fácilmente un Intent para las acciones de "share" que comúnmente utilizamos entre nuestra aplicación y las redes sociales.

Esta clase nos permitirá construir de una manera rápida un Intent para compartir textos, imágenes y archivos de cualquier tipo.

Ejemplo creación de un intent para compartir un texto:

```
Intent shareIntent = ShareCompat.IntentBuilder.from(this)
                        .setType("text/plain")
                        .setText("Texto para compartir")
                        .getIntent();

startActivity(shareIntent);
```

Ejemplo creación de un intent para compartir un archivo:

Primero creamos un archivo y obtenemos la URI (dirección donde se encuentra el recurso) del mismo.

```
try {
    // Creamos el path al archivo
    String filePath = Environment.getExternalStorageDirectory() + "/foo.txt";

    FileWriter fw = new FileWriter(filePath); // Creamos el archivo
    fw.write("LSL UTN-FRA"); // Escribimos un texto en el archivo
    fw.close();

    Uri uri = Uri.fromFile(new File(filePath)); // Creamos la URI al archivo

} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Ahora podemos crear el intent indicando la URI al archivo, de esta forma con dicha información la aplicación que ejecutará el intent podrá acceder al mismo:

```
Intent shareIntent = ShareCompat.IntentBuilder.from(this)
                    .setType("text/plain")
                    .setStream(uri)
                    .getIntent();

startActivity(shareIntent);
```