Actividades e Intenciones

Profesor: Ana Isabel Vegas

INDICE

1. INTRODUCCIÓN

Una actividad (Activity) lo podemos ver como las pantallas de nuestra aplicación. Como vimos en el modulo anterior una actividad se divide en dos ficheros:

- La clase Java donde detallamos el modelo de la actividad
- Un documento xml donde diseñamos la pantalla.

Muchas veces es necesario arrancar una actividad desde otra actividad. Para que esta acción sea posible necesitamos crear un intento (Intent).

A lo largo de este modulo también veremos cómo enviar un mensaje a otra parte de la aplicación o incluso a otra aplicación a través de difusiones y por supuesto a como recibirlas con los receptores de difusiones (BroadcastReceiver).

2.- ACTIVIDADES

CREAR ACTIVIDADES

Para crear una actividad la clase debe heredar de Activity. Esta clase define una serie de métodos para poder controlar el ciclo de vida de una actividad.

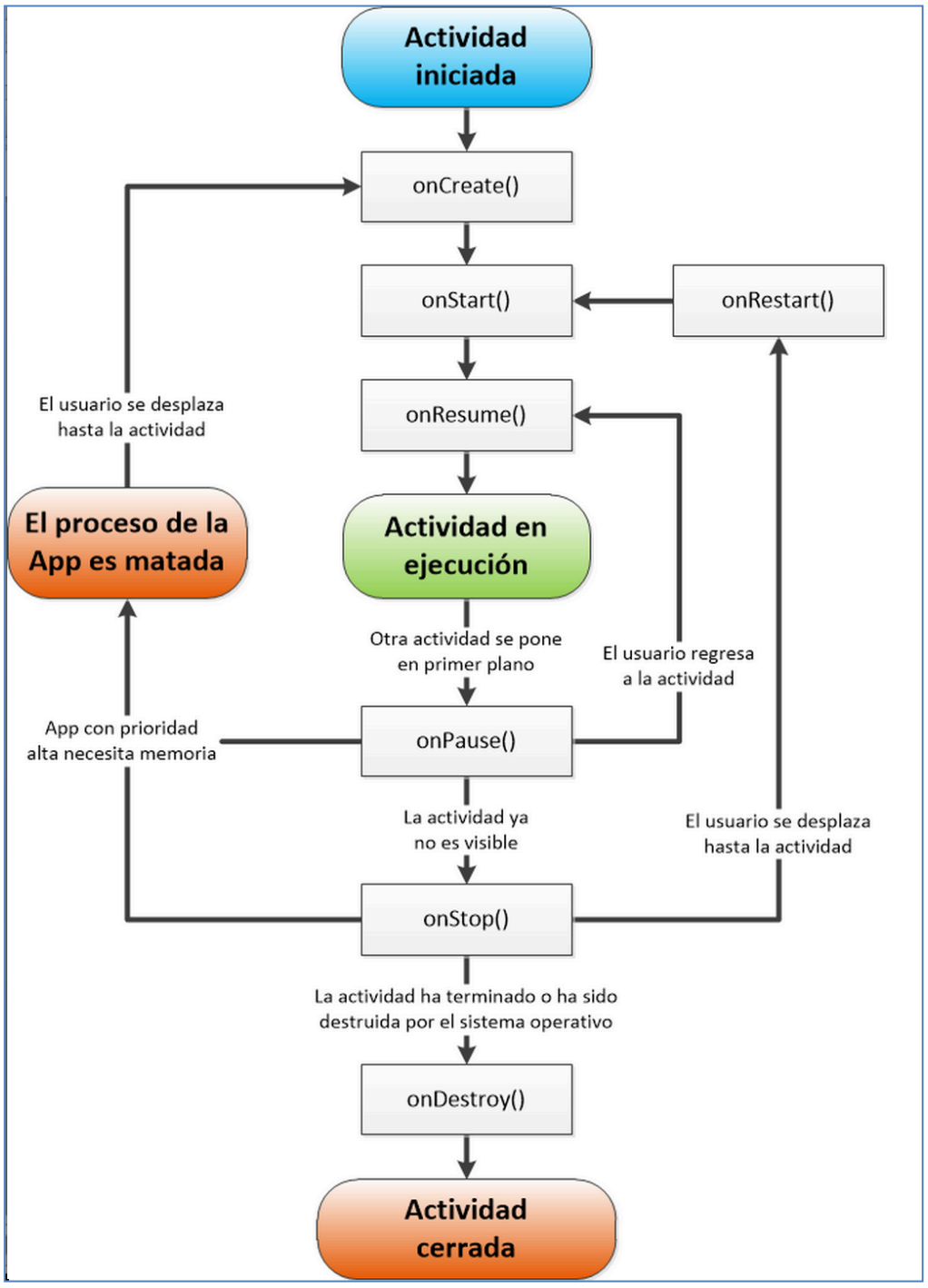

Gráfico 1. Ciclo de vida de las actividades

Estos métodos los podemos ver como manejadores de eventos que se invocan como respuesta a un evento ocurrido en la actividad.

Los más importantes son:

- onCreate(); Se invoca cuando la actividad es creada por primera vez
- onStart(); Se invoca cuando la actividad se muestra al usuario por primera vez.
- onResume(); Se invoca cuando el usuario comienza a interactuar con la actividad.
- onPause(); Ese método es llamado cuando la actividad es pausada y otra actividad previa interactúa con el usuario. Imaginemos que desde la actividad1 se arranca la actividad2. Cuando el usuario deja de utilizar la actividad2 y retorna a la actividad1, la actvidad2 se queda pausada y la actividad1 lanza el evento resume.
- onStop(); Se ejecutará cuando la actividad deja de ser visible para el usuario.
- onDestroy(); Se invoca justo antes de que el sistema destruya la actividad. Una actividad puede ser destruida manualmente o lo puede hacer el sistema para preservar la memoria.
- onRestart(); Se ejecuta cuando la actividad ha sido parada (stop) y se inicia de nuevo.

En los siguientes gráficos vemos como se crea una actividad:

- MainActivity.java
- activity_main.xml

```java
package com.example.helloworld;

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

public class MainActivity extends Activity {

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


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

Gráfico 2. MainActivity.java

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

Gráfico 3. activity_main.xml

Una vez creada la actividad esta debe ser declarada en el archivo AndroidManifest.xml.

```xml
<activity
    android:name="com.example.helloworld.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>
```

Gráfico 4. AndroidManifest.xml

RECUERDA QUE...

- Toda actividad está condicionada por los eventos generados en su ciclo de vida.
- La actividad debe registrarse en el archivo AndroidManifest.xml para que la aplicación tenga en cuenta su existencia.

3.- INTENCIONES

Una intención representa la voluntad de realizar alguna acción o tarea como realizar una llamada de teléfono o visualizar una página web.

INTENCIONES IMPLICITAS Y EXPLICITAS

Podemos clasificar las intenciones como:

- Intenciones explícitas; se indica exactamente el componente a lanzar.
- Intenciones implícitas; se solicitan tareas que no son concretas como "quiero enviar un mail", "quiero mostrar una página web".

CATEGORIAS DE LAS INTENCIONES

Las categorías sirven para indicar el tipo de componente que ha de ser lanzado. Podremos generar nuestras propias categorías pero la clase Intent define unas categorías genéricas que podemos utilizar:

- CATEGORY_BROWSABLE; La actividad lanzada puede ser con seguridad invocada por el navegador para mostrar los datos referenciados por un enlace - por ejemplo, una imagen o un mensaje de correo electrónico.
- CATEGORY_GADGET; La actividad puede ser embebida dentro de otra actividad que aloja gadgets.
- CATEGORY_HOME; La actividad muestra la pantalla de inicio, la primera pantalla que ve el usuario cuando el dispositivo está encendido o cuando la tecla HOME es presionada.
- CATEGORY_LAUNCHER; La actividad puede ser la actividad inicial de una tarea y se muestra en el lanzador de aplicaciones de nivel superior.
- CATEGORY_PREFERENCE; La actividad a lanzar es un panel de preferencias.

COMUNICACION ENTRE ACTIVIDADES

Supongamos que nuestra aplicación tiene dos actividades que por supuesto han de estar declaradas en el archivo AndroidManifest.xml.

```
<activity
    android:name="com.example.vincularactividades.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>

<activity
    android:name=".Activity2"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="com.example.vincularactividades" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Gráfico 5. Declaración de dos actividades

Supongamos que nos encontramos en la actividad MainActivity y queremos establecer un enlace con la actividad MainActivity2.

Para ello creamos un objeto Intent y le pasamos el nombre de la actividad a conectar tal como se ha definido en el elemento <action> del archivo AndroidManifest.xml

```
// Arrancamos la segunda actividad
Intent i = new Intent("com.example.vincularactividades.Activity2");
startActivity(i);
```

Gráfico 6. Enlace a Activity2

Otra forma de hacerlo es crear un objeto Intent y llamar a su método setAction() para definir el nombre de la actividad con la que queremos enlazar.

Esta forma se utiliza para llamar a una actividad que se encuentra dentro de la misma aplicación, aunque también se puede utilizar para que otras aplicaciones llamen a nuestra actividad.

```
// Arrancamos la segunda actividad
   Intent i = new Intent();
   i.setAction("com.example.vincularactividades.Activity2");
   startActivity(i);
```

Gráfico 7. Enlace a Activity2 en la misma aplicación

Cuando queremos llamar a una actividad que se encuentre dentro de la aplicación también podemos utilizar el nombre de su clase.

```
// Arrancamos la segunda actividad
Intent i = new Intent(this, Activity2.class);
startActivity(i);
```

Gráfico 8. Enlace a Activity2 con el nombre de la clase

Si queremos evitar que otras actividades llamen a nuestra actividad desde fuera de la aplicación bastará con eliminar el elemento <action> que se encuentra dentro del elemento <intent-filter>

```
<activity
    android:name=".Activity2"
    android:label="@string/app_name" >
    <intent-filter>
      <!--  <action android:name="com.example.vincularactividades" />   -->
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Gráfico 9. Impedir que se llame a nuestra aplicación

Si la actividad que pretendemos llamar no existe en el dispositivo, la aplicación fallaría y mostraría un mensaje de error avisando que se detiene la aplicación.

Para evitar esto tendremos que llamar al método Intent.createChooser(). Este método toma un objeto Intent y lo utiliza para mostrar una cadena de texto cuando no se localice la actividad o cuando haya varias más que tengan ese nombre.

```
// No localizamos la actividad
Intent i = new Intent("com.example.vincularactividades.Activity3");
startActivity(Intent.createChooser(i, "Choose an application"));
```

Gráfico 10. Mostrar mensaje si no encuentra la actividad

Debemos pasar el nombre completo de la actividad, no el nombre de la clase. El siguiente código no funciona.

```
// Este codigo nunca arrancara la actividad 2
Intent i = new Intent(this, Activity2.class);
startActivity(Intent.createChooser(i, "Choose an application"));
```

Gráfico 11. Error al pasar el nombre de la clase

INTERCAMBIAR DATOS ENTRE ACTIVIDADES

Para pasar información a otra actividad también podemos utilizar la clase Intent.

Para pasar los tipos de datos primitivos a otra actividad utilizaremos el método putExtra().

```
public void onClick(View view) {
    Intent i = new
            Intent("com.example.intercambiardatos.SecondActivity");

    //---pasamos los datos como clave/valor
    i.putExtra("str1", "Esto es un texto");
    i.putExtra("age1", 25);
```

Gráfico 12. Paso de datos String e int

También podemos utilizar el objeto Intent para pasar un objeto Bundle.

```
//----  utilizamos el objeto Bundle para añadir pares clave/valor
Bundle extras = new Bundle();
extras.putString("str2", "Otro texto");
extras.putInt("age2", 35);
```

Gráfico 13. Pasar datos en un objeto Bundle

Para llamar a otra actividad con intención de obtener datos de ella utilizaremos el método startActivityForResult()

```
//---arrancamos la actividad de forma que recojamos el resultado devuelto
startActivityForResult(i, 1);
```

Gráfico 14. Arrancamos la actividad esperando un resultado

El código pasado, en el ejemplo 1, ha de ser una valor mayor o igual que cero. Se utilizará para identificar las actividades que se obtengan, porque podemos llamar simultáneamente a varias actividades.

Si el valor del código solicitado fuese -1, la llamada a startActivityForResult() sería equivalente startActivity(). Es decir, no podría recuperar los datos que devolviese la actividad.

Para que la actividad arrancada fuese capaz de recuperar datos, deberíamos utilizar el método getIntent() para obtener una instancia del objeto Intent que contendrá los datos que ha de recibir. En el caso de recibir datos simples o primitivos utilizamos el método get<type>Extra() donde type puede ser String, int, float.

```
//---get the data passed in using getStringExtra()---
Toast.makeText(this,getIntent().getStringExtra("str1"),
    Toast.LENGTH_SHORT).show();

//---get the data passed in using getIntExtra()---
Toast.makeText(this,Integer.toString(
    getIntent().getIntExtra("age1", 0)),
    Toast.LENGTH_SHORT).show();
```

Gráfico 15. Recuperar los datos enviados

Para recuperar los datos que se han entregado a través del objeto Bunde, utilizamos el método getExtras() del objeto Intent donde utilizamos el método get<type>().

```
//---get the Bundle object passed in---
Bundle bundle = getIntent().getExtras();

//---get the data using the getString()---
Toast.makeText(this, bundle.getString("str2"),
    Toast.LENGTH_SHORT).show();

//---get the data using the getInt() method---
Toast.makeText(this,Integer.toString(bundle.getInt("age2")),
    Toast.LENGTH_SHORT).show();
```

Gráfico 16. Recuperar los datos de un objeto Bundle

La actividad de destino también puede devolver datos a la actividad que realiza la llamada. Para conseguirlo, crearemos otro objeto Intent y definiremos los valores tal y como describimos anteriormente.

```
public void onClick(View view) {
    //---use an Intent object to return data---
    Intent i = new Intent();

    //---use the putExtra() method to return some
    // value---
    i.putExtra("age3", 45);
```

Gráfico 17. Crear el intent para devolver datos

También podemos utilizar el método setData() para entregar un objeto Uri a través de un objeto Intent.

```
//---use the setData() method to return some value---
i.setData(Uri.parse(
    "http://www.grupoatrium.com"));
```

Gráfico 18. Devolver un objeto Uri

Para devolver el resultado a la actividad que realiza la llamada, utilizaremos el método setResult().

```
//---set the result with OK and the Intent object---
setResult(RESULT_OK, i);
```

Gráfico 19. Devolvemos el resultado

La constante RESULT_OK indica a la actividad que realiza la llamada si debe ignorar los datos.

Si queremos que la actividad los ignore, utilizaremos la constante RESULT_CANCELLED. La única responsable de como se interpretarán los resultados será la propia actividad que realiza la llamada, pero el uso de estas constantes le servirá de indicación.

En la actividad principal implementamos el método onActivityResult() para recoger los datos recibidos. Aquí comprobaremos el código de solicitud para asegurarnos de que el resultado obtenido se correspondo con dicha actividad. El código de solicitud será el número que pasamos anteriormente utilizando el método startActivityForResult().

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    // ---comprobamos que el codigo es 1
    if (requestCode == 1) {

        // ---si el resultado es OK
        if (resultCode == RESULT_OK) {

            // ---recogemos el resultado con getIntExtra y lo mostramos
            Toast.makeText(this,
                    Integer.toString(data.getIntExtra("age3", 0)),
                    Toast.LENGTH_SHORT).show();

            // ---recogemos el resultado con getData y lo mostramos
            Uri url = data.getData();
            Toast.makeText(this, url.toString(), Toast.LENGTH_SHORT).show();
        }
    }
}
```

Gráfico 20. Procesamos los datos obtenidos

PASAR OBJETOS ENTRE ACTIVIDADES

Aparte de pasar tipos de datos simples o primitivos, los métodos putExtra() y putExtras) también se pueden utilizar para entregar objetos apoyándonos en un objeto Intent.

Los objetos han de cumplir con la restricción de que sean Serializables.

```
public class MyCustomClass implements Serializable {
    private static final long serialVersionUID = 1L;
    String name;
    String email;

    public void setName(String name) {
        this.name = name;
    }

    public String Name() {
        return name;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String Email() {
        return email;
    }
}
```

Gráfico 21. Clase Serializable

Creamos la instancia serializable, le adjuntamos los datos y lo agregamos al Intent con el método putExtra().

```
// ---creamos nuestro objeto personalizado
MyCustomClass myObject = new MyCustomClass();
myObject.setName("Atrium");
myObject.setEmail("administracion@grupoatrium.com");
i.putExtra("MyObject", myObject);
```

Gráfico 22. Crear el objeto personalizado

Para recuperar el objeto que se ha entregado a otra actividad, utilizaremos el metodo getSerializableExtra() del objeto Intent y le entregaremos la clave que definimos anteriormente con el método putExtra().

```
//---get the custom object passed in---
MyCustomClass obj = (MyCustomClass)
                    getIntent().getSerializableExtra("MyObject");
Toast.makeText(this, obj.Name(), Toast.LENGTH_SHORT).show();
Toast.makeText(this, obj.Email(), Toast.LENGTH_SHORT).show();
```

Gráfico 23. Recuperar el objeto pasado a la actividad

RECUERDA QUE...

- Podemos vincular actividades a través de las intenciones (Intent).
- A través de un Intent podemos pasar datos simples y objetos.
- Los objetos a transferir deben ser Serializables

4.- DIFUSIONES

Una difusión nos permite enviar un mensaje a otra parte de la aplicación o incluso a otra aplicación. Para ello necesitamos tener un receptor que escuche todas las difusiones que tengan lugar.

PROGRAMAR UN RECEPTOR DE DIFUSIONES

Debemos crear una clase que herede de BroadCastReceiver y sobreescribir el método onReceive() detallando que queremos hacer con la difusión recibida.

```
public class MyBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent i) {
        Toast.makeText(context,
                "Received broadcast in MyBroadcastReceiver, value received: " +
                 i.getStringExtra("key"),
                Toast.LENGTH_LONG).show();
    }
}
```

Gráfico 24. Crear un BroadcastReceiver en la actividad.

Para probarlo debemos enviar una difusión.

Para obtener los datos que se han de entregar al receptor, utilizaremos un objeto Intent que entregaremos como segundo argumento al método onReceive(). Para utilizar esta clase, debemos crear una instancia suya y un objeto IntentFilter:

```
public class MainActivity extends Activity {
    MyBroadcastReceiver myReceiver;
    IntentFilter intentFilter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        myReceiver = new MyBroadcastReceiver();
        intentFilter = new IntentFilter("MY_SPECIFIC_ACTION");
    }
```

Gráfico 25. Crear el objeto BroadcastReceiver

En el constructor del objeto IntentFilter especificamos la acción definida por el usuario utilizando nuestra propia cadena.

Para registrar el objeto BroadcastReceiver, utilizamos el método registerReceiver) y se lo pasamos al objeto BroadcastReceiver junto con el objeto IntentFilter:

```
registerReceiver(myReceiver, intentFilter);
```

Gráfico 26. Registrar el objeto BroadcastReceiver

Ahora que tenemos registrado un objeto BroadcastReceiver, podemos enviar una difusión para comprobar que funciona correctamente. Para ello, pasaremos el método sendBroadcast() a un objeto Intent:

```
public void onClick(View view) {
    Intent i = new Intent("MY_SPECIFIC_ACTION");
    i.putExtra("key", "some value from intent");
    sendBroadcast(i);
```

Gráfico 27. Enviar la difusión

Si quisiésemos entregar datos al receptor, tendriamos que utilizar el metodo putExtra(). Para eliminar el registro del receptor de difusión, se utiliza el metodo unregisterReceiver():

```
unregisterReceiver(myReceiver);
```

Gráfico 28. Eliminar el registro del receptor de difusión

REGISTRAR BROADCASTRECEIVER EN EL ARCHIVO ANDROIDMANIFEST.XML

En el ejemplo anterior, el receptor no funcionará cuando la aplicación se ejecute en segundo plano. En estos casos, el sistema elimina el registro del receptor:

```
@Override
public void onPause() {
    super.onPause();
    //---unregister the receiver---
    unregisterReceiver(myReceiver);
}
```

Gráfico 29. Eliminar el registro del receptor de difusión al pausar la aplicación

Si queremos ue nuestros receptores sean persistentes, tendremos que registrar la clase BroadcastReceiver en el archivo AndroidManifest.xml.

Para ello creamos la clase BroadcastReceiver en otra clase Java.

```
public class MySecondBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent i) {
        Toast.makeText(context,
            "Received broadcast in MySecondBroadcastReceiver; value received: "
                        + i.getStringExtra("key"),
                        Toast.LENGTH_LONG).show();
        //---abort the broadcast----
        abortBroadcast();
    }
}
```

Gráfico 30. Crear el BroadcastReceiver en otra clase

Para registrar este receptor en el archivo AndroidManifest.xml añadimos el elemento <receiver>

```
<receiver android:name=".MySecondBroadcastReceiver" >
    <intent-filter android:priority="50">
        <action android:name="MY_SPECIFIC_ACTION" />
    </intent-filter>
</receiver>
```

Gráfico 31. Registrar el receptor en AndroidManifest.xml

ASIGNAR PRIORIDADES A LOS RECEPTORES DE DIFUSIÓN

Para asignar prioridades a los receptores por medio de la programación, utilizaremos el método setPriority().

```
intentFilter.setPriority(10);
```

Gráfico 32. Asignar prioridades

El método setPriority() trabaja con un valor comprendido entre 0 (valor predeterminado) y 1000. Cuanto mayor sea este número, mayor será la prioridad asignada. La aplicación llamará antes a los receptores con la prioridad más alta.

Si hubiese varios receptores que tuviesen la misma prioridad, se les llamaría de forma aleatoria.

Para definir la prioridad de los receptores en el archivo AndroidManifest.xml utilizaremos el atributo android:priority:

```
<receiver android:name=".MySecondBroadcastReceiver" >
    <intent-filter android:priority="50">
        <action android:name="MY_SPECIFIC_ACTION" />
    </intent-filter>
</receiver>
```

Gráfico 33. Asignar la prioridad al receptor en AndroidManifest.xml

No podemos utilizar el método sendBroadcast() para enviar una difusión a los receptores con mayor prioridad.

Tenemos que utilizar el método sendOrderedBroadcast(), entregar un objeto Intent y los permisos que deban cumplir los receptores para poder recibir la difusión.

```
public void onClick(View view) {
    Intent i = new Intent("MY_SPECIFIC_ACTION");
    i.putExtra("key", "some value from intent");
    //sendBroadcast(i);

    //---permite cancelar la difusion---
    //---permite que los receptores definan la prioridad---
    sendOrderedBroadcast(i, null);
}
```

Gráfico 34. Envío de difusión a los receptores de mayor prioridad

Si solo quisiéramos enviar una difusión a los receptores que tengan permiso para acceder a Internet, tendríamos que especificar el permiso en el segundo argumento del método sendOrderedBroadcast()

```
sendOrderedBroadcast(i, "android.permission.INTERNET");
```

Gráfico 35. Especificar permiso

CANCELAR UNA DIFUSIÓN

Cuando el elemento que emite la difusión utilice el método sendOrderedBroadcast(), seguirán llamando a los receptores según la prioridad

que tengan asignada. Cuando uno de los receptores con prioridad alta reciba la difusión, se encargará de transmitirla al siguiente receptor de la lista.

En algunos casos, queremos hacernos cargo de la difusión y evitar que se siga propagando a otros receptores. Para ello, utilizaremos el método abortBroadcast()

```
public class MySecondBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent i) {
        Toast.makeText(context,
            "Received broadcast in MySecondBroadcastReceiver; value received: "
                        + i.getStringExtra("key"),
                        Toast.LENGTH_LONG).show();

        //---cancela la difusion----
        abortBroadcast();
    }
}
```

Gráfico 36. Cancelar una difusión

Para cancelar una difusión con el método abortBroadcast(), tendremos que enviarla con el método sendOrderedBroadcast(). El método sendBroadcast() no tiene ningún efecto sobre la prioridad y no permite cancelar la difusión.

RECUERDA QUE...

- Los receptores de difusiones se pueden programar dentro de la actividad o en una clase aparte
- Podemos establecer prioridades a los receptores
- Para cancelar una difusión con el método abortBroadcast(), tendremos que enviarla con el método sendOrderedBroadcast(). El método sendBroadcast() no tiene ningún efecto sobre la prioridad y no permite cancelar la difusión.

ÍNDICE DE GRÁFICOS