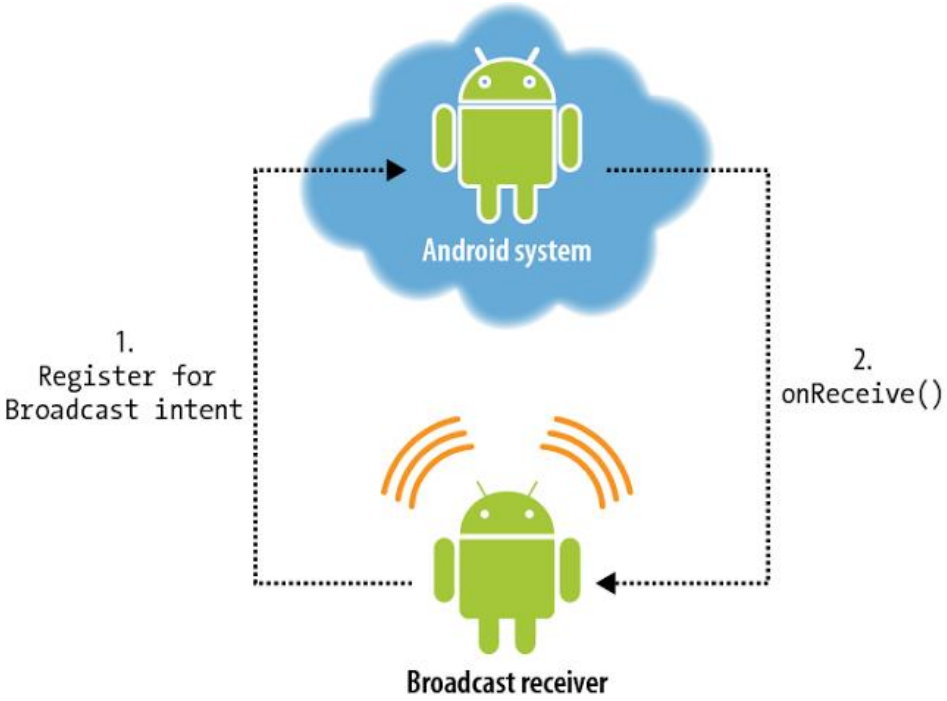
Broadcast Receiver



Contenidos

Componentes de una aplicación

- Activity √
- Intent √
- Content Provider √
- BroadCast Receiver
- Service

Broadcast receiver - Receptores de avisos/anuncios/emisiones

Un Broadcast Receiver es el componente que está destinado a recibir y responder a:

- eventos globales generados por el sistema (como un aviso de batería baja, de SMS recibido, de una llamada, etc.)
- eventos producidos por otras aplicaciones (como, por ejemplo, un aviso de que el consumo de datos ha llegado a un límite, etc.).

Aunque un Broadcast Receiver no se carga en memoria inicialmente, está a la espera de algún evento para ejecutarse.

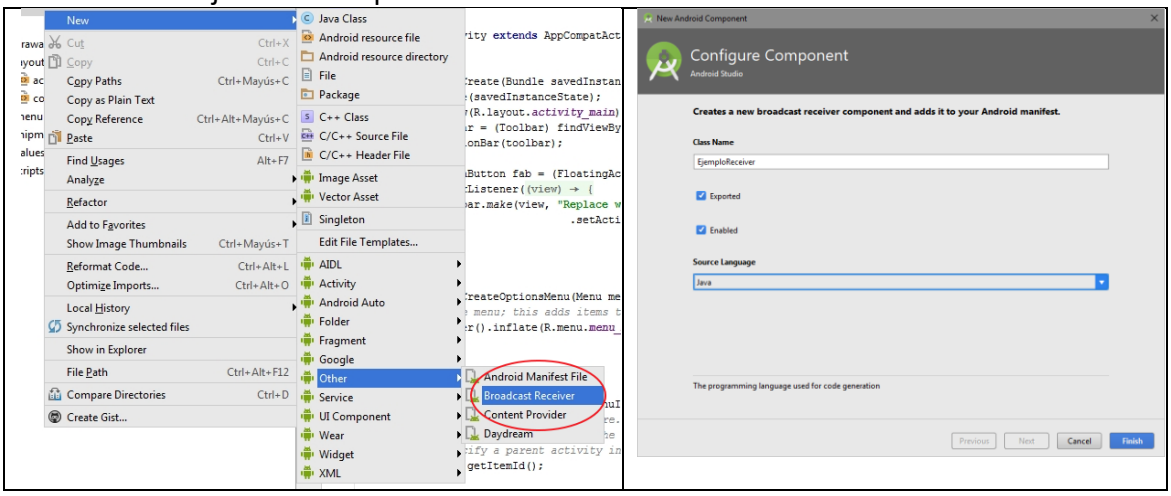
Los receptores de anuncios no tienen interfaz de usuario, aunque pueden iniciar una actividad (no aconsejable, ya que pueden irritar al usuario al interrumpir lo que estaba haciendo con otra aplicación) o responder lanzando una notificación para informar al usuario o lanzando un servicio (siguiente tema).

Desde Android 3.1, el sistema excluye por defecto a todo BroadcastReceiver poder recibir eventos si la correspondiente aplicación que lo registra nunca ha sido iniciada por el usuario o si el usuario explícitamente ha detenido la aplicación desde el menú de ajustes de aplicaciones. Es una característica de seguridad adicional ya que el usuario puede estar seguro de que sólo las aplicaciones que él empezó recibirán broadcast Intents. Esto no quiere decir que la aplicación tenga que estar en marcha, significa que alguna vez el usuario la ha iniciado! Incluso el receptor de anuncios funcionará después de un reinicio del dispositivo.

Para usarlos debemos crear una clase que extienda de de BroadcastReceiver con el ciclo de vida de lo que debe hacer y declararlo en el AndroidManifest.

Usando el asistente de creación de Broadcast Receiver de AS

Parte del trabajo nos lo simplifica AS:



Nos ha creado la clase y la declaración:

```java
public class EjemploReceiver extends
BroadcastReceiver {
    public EjemploReceiver() {
    }

    @Override
    public void onReceive(Context
context, Intent intent) {
        // TODO: This method is called
when the BroadcastReceiver is receiving
        // an Intent broadcast.
        throw new
UnsupportedOperationException("Not yet
implemented");
    }
}
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest package="com.pdm.p_88_broadcast_1"

xmlns:android="http://schemas.android.com/apk/res/android">

    …

        <receiver
            android:name=".EjemploReceiver"
            android:enabled="true"
            android:exported="true">
        </receiver>
    </application>

</manifest>
```

Declaración del componente en AndroidManifest

android:enabled = ["true | false"] activa / desactiva el receptor.
android:exported = ["true | false"] recibimos anuncios desde otras aplicaciones/solo recibimos anuncios emitidos desde otros componentes de nuestra aplicación.

Ciclo de vida

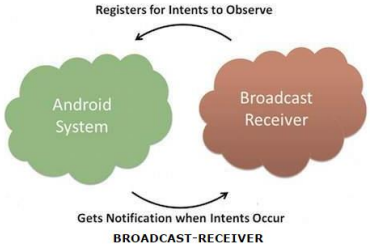
El ciclo de vida de un BroadcastReceiver es muy sencillo, solo dispone del método onReceive() donde escribiremos el código que queramos que se ejecute cuando se produzca el evento en cuestión, teniendo en cuenta que:

- Mientras está ejecutándose ese proceso tiene alta prioridad. Debe ser lo más rápido posible ya que es ejecutado por el hilo principal (esto significa que bloquea el manejo de la interfaz de usuario). Como no debe bloquear al sistema, si tiene que realizar una tarea costosa tendrá que lanzar un hilo secundario o mejor llamar a un servicio para que la haga.
- No se debe mostrar un cuadro de diálogo, en su lugar se puede lanzar una notificación.
- Si se desea una acción persistente en el tiempo resulta muy frecuente lanzar un servicio. No se puede unir a un servicio (bindService()) pero sí se puede utilizar startService() para arrancar un servicio.
- Recibe como argumentos:
    - context: se puede utilizar para acceder a información adicional (recursos) o para iniciar servicios o actividades
    - intent: la acción que ha lanzado el broadcast, contiene información adicional a la que acceder mediante el método getExtras()

Un objeto BroadcastReceiver sólo existe durante la llamada a onReceive(). El sistema crea el BroadcastReceiver, llama a este método y cuando termina destruye el objeto (un detalle importante es que no hace falta tener la aplicación en marcha donde se registra el BroadcastReceiver para que se active).

Registrar el evento al que responde el receptor

¿Cómo sabe el sistema que tiene que crear el BroadcastReceiver? Porque la aplicación lo ha registrado fijando ante que anuncio debe responder mediante IntentFilter.



Existen dos maneras de registro:

- estáticamente en el fichero AndroidManifest (no se puede con todos los anuncios del sistema!):

```
<receiver
    android:name=".EjemploReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BATTERY_LOW"/>
    </intent-filter>
</receiver>
```

  Si se registra así, el receptor será llamado cada vez que se produce el evento (aunque la aplicación no esté iniciada, repito!). Esto podría ser lo que se desea o podría ser demasiado a menudo.

o

- dinámicamente (normalmente en el método onResume() de la Activity/Fragment) con el método **registerReceiver()**. Por ejemplo, para cuando quede poca batería:

  registerReceiver(EjemploReceiver,new IntentFilter(Intent.*ACTION_BATTERY_LOW*));

  y se puede "desregistrar" (normalmente en el método onPause())

  unregisterReceiver(**EjemploReceiver**);

  Al hacerlo de esta manera (registrar/desregistrar), el sistema ante el aviso solo creará BroadcatReceiver si la actividad no está en pausa.

¿Qué manera elegir?

Depende de lo que la aplicación hace con el evento recibido:

- Si la aplicación ofrece algún tipo de servicio en torno al evento (por ejemplo, tiene que hacer algo tan pronto como el dispositivo ha arrancado o cada vez que se instala una aplicación, etc.) debe registrarse el BroadcastReceiver en el archivo de manifiesto.
- Si la aplicación debe responder de manera distinta ante cambios de estado (por ejemplo, depende de una conexión Bluetooth establecida), entonces tiene que reaccionar ante un posible cambio de estado pero sólo cuando está activa. En este caso, lo razonable es un registro dinámico.
- También hay una serie de eventos que no se les permite registrar de forma estática. Un ejemplo es el caso Intent.ACTION_TIME_TICK que se emite cada minuto. Un receptor estático podría agotar la batería.

Permisos

Para poder registrar algunos eventos se necesita tener los permisos adecuados.
Por ejemplo, si la aplicación necesita ser avisada de que el inicio del dispositivo se ha completado (Intent.ACTION_BOOT_COMPLETED) en el fichero AndroidManifest hay que fijar:

```
<manifest
    package="com.pdm.ejercicios.broadcast_1"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>

    <application
```

Anuncios del sistema

En la siguiente imagen figura un listado con algunos de los anuncios del sistema (batería baja, llamada telefónica,…) a los que se puede responder. También figura si se puede registrar o no en el Manifest y si solo el sistema puede generar el anuncio.

Recursos adicionales: Algunos anuncios broadcast

Lista de los anuncios broadcast más importantes organizados por temas. (No Manifest): No se puede declarar el receptor de anuncios en AndroidManifest.xml. Solo se puede utilizar `registerReceiver()`. (Solo sistema): Intención protegida que solo puede ser lanzada por el sistema.

Nombre de la acción /(CONSTANTE)	Descripción / Permiso (INFORMACIÓN EXTRA EN INTENT)
Batería	
android.intent.action.BATTERY_LOW (ACTION_BATTERY_LOW)	Batería baja (Solo sistema)
android.intent.action.BATTERY_OKAY (ACTION_BATTERY_OKAY)	Batería correcta después de haber estado baja (Solo sistema)
android.intent.action.ACTION_POWER_CONNECTED (ACTION_POWER_CONNECTED)	La alimentación se ha conectado (Solo sistema)
android.intent.action.ACTION_POWER_DISCONNECTED (ACTION_POWER_DISCONNECTED)	La alimentación se ha desconectado (Solo sistema)
android.intent.action.BATTERY_CHANGED (ACTION_BATTERY_CHANGED)	Cambia el estado de la batería (No Manifest) (Solo sistema)
Sistema	
android.intent.action.BOOT_COMPLETED (ACTION_BOOT_COMPLETED)	sistema operativo cargado. Permiso RECEIVE_BOOT_COMPLETED (Solo sistema)
android.intent.action.ACTION_SHUTDOWN (ACTION_SHUTDOWN)	El dispositivo va a ser desconectado (Solo sistema)
android.intent.action.AIRPLANE_MODE (ACTION_AIRPLANE_MODE_CHANGED)	modo vuelo activo (Solo sistema)
android.intent.action.TIME_TICK (ACTION_TIME_TICK)	Se envía cada minuto. (No Manifest) (Solo sistema)
android.intent.action.TIME_SET (ACTION_TIME_CHANGED)	La fecha/hora es modificada (Solo sistema)
android.intent.action.CONFIGURATION_CHANGED (ACTION_CONFIGURATION_CHANGED)	Cambia la configuración del dispositivo (orientación, idioma,..) (No Manifest) (Solo sistema)

Entradas y pantalla

android.intent.action.SCREEN_OFF (ACTION_SCREEN_OFF)	**La pantalla se apaga** (Solo sistema)
android.intent.action.SCREEN_ON (ACTION_SCREEN_ON)	**La pantalla se enciende** (Solo sistema)
android.intent.action.CAMERA_BUTTON (ACTION_CAMERA_BUTTON)	**Se pulsa el botón de la cámara.** (EXTRA_KEY_EVENT)
android.intent.action.HEADSET_PLUG (ACTION_HEADSET_PLUG)	**Se conectan los auriculares** (extras: state, name, microphone)
android.intent.action.INPUT_METHOD_CHANGED (ACTION_INPUT_METHOD_CHANGED)	**Cambia método de entrada**
android.intent.action.USER_PRESENT (ACTION_USER_PRESENT)	El usuario está presente después de que se active el dispositivo (Solo sistema)

Memoria y Escaner Multimedia

android.intent.action.DEVICE_STORAGE_LOW (ACTION_DEVICE_STORAGE_LOW)	**Queda poca memoria** (Solo Sistema) (Solo sistema)
android.intent.action.DEVICE_STORAGE_OK (ACTION_DEVICE_STORAGE_OK)	**Salimos de la condición de poca memoria** (Solo Sistema) (Solo sistema)
android.intent.action.MEDIA_EJECT (ACTION_MEDIA_EJECT)	**El usuario pide extraer almacenamiento exterior**
android.intent.action.MEDIA_MOUNTED (ACTION_MEDIA_MOUNTED)	**Almacenamiento exterior disponible**
android.intent.action.MEDIA_REMOVED (ACTION_MEDIA_REMOVED)	**Almacenamiento exterior no disponible**
android.intent.action.MEDIA_SCANNER_FINISHED (ACTION_MEDIA_SCANNER_FINISHED)	**El escáner de medios termina un directorio (se indica en Intent.mData)**
android.intent.action.MEDIA_SCANNER_SCAN_FILE (ACTION_MEDIA_SCANNER_SCAN_FILE)	**El escáner de medios encuentra un fichero (se indica en Intent.mData)**
android.intent.action.MEDIA_SCANNER_STARTED (ACTION_MEDIA_SCANNER_STARTED)	**El escáner de medios comienza un directorio (se indica en Intent.mData)**

Aplicaciones

android.intent.action.MY_PACKAGE_REPLACED (ACTION_MY_PACKAGE_REPLACED)	**Una nueva version de tu aplicación ha sido instalada** (Solo sistema)
android.intent.action.PACKAGE_ADDED (ACTION_PACKAGE_ADDED)	**Una nueva aplicación instalado** (EXTRA_UID, EXTRA_REPLACING) (Solo sistema)
android.intent.action.PACKAGE_FIRST_LAUNCH (ACTION_PACKAGE_FIRST_LAUNCH)	**Primera vez que se lanza una aplicación** (Solo sistema)
android.intent.action.PACKAGE_REMOVED (ACTION_PACKAGE_REMOVED)	**Se desinstala una aplicación** (Solo sistema)

Comunicaciones y redes

android.intent.action.PHONE_STATE	llamada de teléfono. Permiso: READ_PHONE_STATE(EXTRA_STATE,EXTRA_STATE_RINGING)
android.intent.action.NEW_OUTGOING_CALL (ACTION_NEW_OUTGOING_CALL)	Se va ha hacer una llamada. Permiso PROCESS_OUTGOING_CALLS(EXTRA_PHONE_NUMBER) (Solo sistema)
android.provider.Telephony.SMS_RECEIVED	SMS recibido. Permiso: RECEIVE_SMS
android.bluetooth.intent.action.DISCOVERY_STARTED	**comienza escáner Bluetooth**
android.bluetooth.intent.action.ENABLED	**Bluetooth habilitado**
android.net.wifi.NETWORK_IDS_CHANGED (NETWORK_IDS_CHANGED_ACTION)	**Cambia la red WiFi**
android.net.wifi.STATE_CHANGE (NETWORK_STATE_CHANGED_ACTION)	**Cambia la conectividad WiFi** (EXTRA_NETWORK_INFO, EXTRA_BSSID, EXTRA_WIFI_INFO)
android.net.wifi.RSSI_CHANGED (RSSI_CHANGED_ACTION)	**Cambia el nivel de señal WiFi** (EXTRA_NEW_RSSI)

Caso práctico 1: Notificaciones para llamadas entrantes

Crea proyecto P_78_BroadCast_1. Utilizando el asistente crearemos el BroadCast llamado EjemploReceiver.

```
public class EjemploReceiver extends BroadcastReceiver {
    Notification.Builder builder;

    @Override
    public void onReceive(Context context, Intent intent) {
        // Nos aseguramos que el intent por el que se ha accedido ha sido el de cambios en el estado
del tfno.
        if (Objects.equals(intent.getAction(), "android.intent.action.PHONE_STATE")) {
            // Sacamos información del intent enviado por el sistema
            Bundle extras = intent.getExtras();
            if (extras != null) {
                String estado = extras.getString(TelephonyManager.EXTRA_STATE);
                if (Objects.equals(estado, TelephonyManager.EXTRA_STATE_RINGING)) {
                    String numero = extras.getString(TelephonyManager.EXTRA_INCOMING_NUMBER);
                    // Creamos Notificación
                    NotificationManager notificationManager = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
                    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
                        String idCanal = "miIdCanal";
                        String nombreCanal = "miNombreCanal";
                        int importancia = NotificationManager.IMPORTANCE_HIGH;
                        NotificationChannel miCanal = new NotificationChannel(idCanal, nombreCanal,
importancia);
                        if (notificationManager != null) {
                            notificationManager.createNotificationChannel(miCanal);
                        }
                        builder = new Notification.Builder(context, idCanal);
                    } else {
                        builder = new Notification.Builder(context);
                        builder.setPriority(Notification.PRIORITY_MAX);
                    }
                    builder.setSmallIcon(android.R.drawable.ic_dialog_alert)
                            .setContentTitle("Llamada entrante")
                            .setContentText(estado + " " + numero)
                            .setAutoCancel(true)
                            .setContentIntent(PendingIntent.getActivity(context, 0, new Intent(),
PendingIntent.FLAG_UPDATE_CURRENT));
                    if (notificationManager != null) {
                        notificationManager.notify(1, builder.build());
                    }
                }
            }
        }
    }
}
```

Permisos y registro en AndroidManifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="com.pdm.p_88_broadcast_1"
          xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>

    <application
        …
        <receiver
            android:name=".EjemploReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.PHONE_STATE"/>
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

Aunque parezca que no hace falta MainActivity, es necesaria para que el usuario pueda arrancar al menos una vez la aplicación que es la medida de seguridad comentada anteriormente.

Además en este ejemplo, como se trata de un permiso peligros (READ_PHONE_STATE) es obligatorio su tratamiento (observa que comparado con otros proyectos, no hay método seguir() ya que realmente no hay que hacer nada):

```
public class MainActivity extends AppCompatActivity {

    private ConstraintLayout layoutMain;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        layoutMain = findViewById(R.id.pantallaPrincipal);
        if (ContextCompat.checkSelfPermission(getApplicationContext(),
Manifest.permission.READ_PHONE_STATE) != PackageManager.PERMISSION_GRANTED) {
            pedirPermiso();
        }
    }

    private void pedirPermiso() {
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
Manifest.permission.READ_PHONE_STATE)) {
            final Activity activity = this;
            Snackbar.make(layoutMain, "Sin el permiso no puedo seguir.", Snackbar.LENGTH_INDEFINITE)
                    .setAction("OK", new View.OnClickListener() {
                        @Override
                        public void onClick(View view) {
                            ActivityCompat.requestPermissions(activity, new
String[]{Manifest.permission.READ_PHONE_STATE}, 123);
                        }
                    })
                    .show();
        } else {
            ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.READ_PHONE_STATE}, 123);
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull
int[] grantResults) {
        if (requestCode == 123) {
            if (grantResults.length == 1 && grantResults[0] != PackageManager.PERMISSION_GRANTED) {
                Snackbar.make(layoutMain, "Sin el permiso, no puedo realizar la acción",
Snackbar.LENGTH_LONG).show();
            }
        }
    }

    …
}
```

Caso práctico 2: Controlar cuando se conecta o desconecta el USB

Un mismo Broadcast Receiver puede responder a distintos eventos.

Por comodidad, no creamos un nuevo proyecto, a nuestro Broadcast_1 le añadimos el Broadcast Receiver de nombre Receiver2:

```
<receiver
    android:name=".Receiver2"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.ACTION_POWER_CONNECTED" />
        <action android:name="android.intent.action.ACTION_POWER_DISCONNECTED" />
    </intent-filter>
</receiver>
```

En el método onReceive() si es necesario podemos obtener la acción desencadenante y obrar en consecuencia:

```
public void onReceive(Context context, Intent intent) {
    String motivo = intent.getAction();
    if (motivo.equals(Intent.ACTION_POWER_CONNECTED)) {
        Toast.makeText(context, "Conectado", Toast.LENGTH_SHORT).show();
    }
    else {
        Toast.makeText(context, "Desconectado", Toast.LENGTH_SHORT).show();
    }
}
```

Caso práctico 3: Arrancar una actividad al arrancar el dispositivo

Crea una nueva actividad (SegundaActivity) con la imagen que deseas en su layout (ic_launcher, para no perder tiempo) y añadimos el broadcast llamado Receiver3:

```
public void onReceive(Context context, Intent intent) {
    if (Objects.equals(intent.getAction(), Intent.ACTION_BOOT_COMPLETED)) {
        Intent miIntent = new Intent(context, SegundaActivity.class);
        miIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        context.startActivity(miIntent);
    }
}
```

En el AndroidManifest:

```
<manifest package="com.pdm.p_88_broadcast_1"
          xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />

    <application
        …

        <receiver
            android:name=".Receiver3"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
```

Al reiniciar el dispositivo, puede que te parezca que la imagen tarda en aparecer, pero aparece!

(Quizás, para no irritar al usuario, sería mejor lanzar una notificación y al clickar sobre ella que se abra la SegundaActivity!).

Los Broadcast Receiver son muchas veces utilizados para lanzar servicios (siguiente tema) al reiniciar el dispositivo.

Observa que en nuestros ejemplos a veces hemos puesto:

```
if (Objects.equals(intent.getAction(), "android.intent.action.PHONE_STATE")) {
```

y otras

```
if (Objects.equals(intent.getAction(), Intent.ACTION_BOOT_COMPLETED)) {
```

es porque hemos hecho uso de las siguientes constantes declaradas:

- ACTION_TIME_TICK
- ACTION_TIME_CHANGED
- ACTION_TIMEZONE_CHANGED
- ACTION_BOOT_COMPLETED
- ACTION_PACKAGE_ADDED
- ACTION_PACKAGE_CHANGED
- ACTION_PACKAGE_REMOVED
- ACTION_PACKAGE_RESTARTED
- ACTION_PACKAGE_DATA_CLEARED
- ACTION_PACKAGES_SUSPENDED
- ACTION_PACKAGES_UNSUSPENDED
- ACTION_UID_REMOVED
- ACTION_BATTERY_CHANGED
- ACTION_POWER_CONNECTED
- ACTION_POWER_DISCONNECTED
- ACTION_SHUTDOWN

Caso práctico 4: Código oculto

Son los códigos que al introducirlos en el teclado del "teléfono" (el de llamar!) permiten acceder a alguna función especial, como ver el IMEI (*#06#) o información del teléfono (*#*#4636#*#*).

Uso: *#*#código#*#*

Más códigos ocultos (ten cuidado con algunos!)

Añadimos un nuevo Broadcast Receiver de nombre Receiver4:

```
@Override
public void onReceive(Context context, Intent intent) {
    if (intent.getAction().equals("android.provider.Telephony.SECRET_CODE")) {
        String numero = intent.getData().getHost();
        if (numero.equals("732")) {
            Toast.makeText(context, "¡Función chachipiruli altamente secreta desbloqueada!",
Toast.LENGTH_SHORT).show();
        }
    }
}
```

AndroidManifest:

```
<receiver
    android:name=".Receiver4"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.provider.Telephony.SECRET_CODE"/>
        <data
            android:host="732"
            android:scheme="android_secret_code"/>
    </intent-filter>
</receiver>
```

(Aunque no tiene que ver con el tema, otro huevo de Pascua de Android es: En Ajustes, Acerca del teléfono, pulsa varias veces en Version de Android: aparece una animación).

Caso Práctico 5: Toast de aviso de WIFI si/no

```
public class MainActivity extends AppCompatActivity {
    …
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        …
        miReceiver = new Receiver5();
        IntentFilter filter = new IntentFilter("android.net.conn.CONNECTIVITY_CHANGE");
        registerReceiver(miReceiver, filter);
    }
    …
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Desregistrar
        unregisterReceiver(miReceiver);
    }
    …

public class Receiver5 extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (Objects.equals(action, "android.net.conn.CONNECTIVITY_CHANGE")) {
            ConnectivityManager conMgr = (ConnectivityManager)
context.getSystemService(Context.CONNECTIVITY_SERVICE);
            if (conMgr!=null && conMgr.getActiveNetworkInfo() != null &&
                    conMgr.getActiveNetworkInfo().isAvailable() &&
                    conMgr.getActiveNetworkInfo().isConnected()) {
                int tipo = conMgr.getActiveNetworkInfo().getType();
                if (tipo == ConnectivityManager.TYPE_WIFI)
                    Toast.makeText(context, "WIFI SI", Toast.LENGTH_LONG).show();
                else
                    Toast.makeText(context, "WIFI NO", Toast.LENGTH_LONG).show();
            } else
                Toast.makeText(context, "Sin acceso a conexión", Toast.LENGTH_LONG).show();
        }
    }
}
```

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Anuncios propios

Además de los del sistema, nuestras aplicaciones pueden enviar anuncios para los que otras aplicaciones hayan definido Broadcast Receiver (el proceso es similar a como tratamos a los intents implícitos).

El método sendBroadcast() de la clase Context permite enviar BroadcastIntents. No se pueden desencadenar emisiones del sistema, el sistema Android evitará esto. Pero sí podemos definir intent-filters para nuestras propias acciones y hacerlas funcionar desde el método sendBroadcast().

Enviar anuncios a nuestra propia aplicación

```
<receiver
    android:name=".ReceiverPropio"
    android:enabled="true"
    android:exported="false">
</receiver>
```

La clase LocalBroadcastManager se utiliza para registrar y enviar anuncios locales dentro del propio proceso. Es más rápido y más seguro ya que los eventos no salen de la aplicación.

```
public class MainActivity extends AppCompatActivity {

    private ReceiverPropio miReceiver2;
    ...
    @Override
    public void onResume() {
        super.onResume();
        // Registrar
        miReceiver2 = new ReceiverPropio();
        LocalBroadcastManager.getInstance(this).registerReceiver(miReceiver2, new
IntentFilter("miAnuncio"));
        //Cuando pase algo, no como aqui que lo hace nada más iniciarse la actividad
        Intent intent = new Intent();
        intent.setAction("miAnuncio");
        intent.putExtra("message", "datoEnviado");
        LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
    }

    @Override
    protected void onPause() {
        super.onPause();
        // Desregistrar
        LocalBroadcastManager.getInstance(this).unregisterReceiver(miReceiver2);
        super.onPause();
    }
    ...

public class ReceiverPropio extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        String message = intent.getStringExtra("message");
        // lo que haya que hacer
    }
}
```

Enviar anuncios a otras aplicaciones

La aplicación que quiere enviar el anuncio:

```
Intent intent = new Intent();
intent.setAction("miAnuncio");
intent.putExtra("message", "datoEnviado");
context.sendBroadcast(intent);
```

Si solo se desea que "se entere" una determinada aplicación y no todas las que tengan registros para ese anuncio, se le añade al intent el paquete de la aplicación deseada:

```
intent.setPackage("paquete.deseado");
```

Incluso se pueden fijar permisos que deben tener las aplicaciones que los traten:

```
sendBroadcast (intent, stringDePermisos);
```

Las aplicaciones que deben recoger el anuncio deben tener registrado un Broadcast Receiver que lo trate bien desde AndroidManifest o dinámicamente:

```
intentFilter = new IntentFilter("android.intent.action.NOMBRE_DEL_INTENT");
registerReceiver(mReceiver, intentFilter);
```

Activar/desactivar broadcast registrados estáticamente.

Al registrar en AndroidManifest hemos dejado el atributo

```
android:enabled="true"
```

si lo ponemos con valor "false" podremos habilitarlo/deshabilitarlo desde código:

```
ComponentName receiver = new ComponentName(context, myReceiver.class);
PackageManager pm = context.getPackageManager();
pm.setComponentEnabledSetting(receiver, PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
PackageManager.DONT_KILL_APP);
```

Por defecto, PackageManager mata a la aplicación inmediatamente, ya que un cambio de estado de los componentes podría dar lugar a situaciones impredecibles. La bandera DONT_KILL_APP evita que esto suceda y es segura cuando se usa para BroadcastReceivers.

Observaciones

En el código de EjemploReceiver, para lanzar la notificación hemos usado PendingIntent.getActivity(), pero no debería ser PendingIntent.getBroadcast()?
El intent que queremos es new Intent(), es decir null y por tanto da lo mismo en este caso.
En general, si el intent es para lanzar una nueva Activity debe ser PendingIntent.getActivity(), si lo es para lanzar un nuevo broadcast-receiver debe ser PendingIntent.getBroadcast().

Usos maliciosos de los Broadcast que seguro que te interesan:

Broadcast Receivers, o cómo averiguar qué trama tu novia en su móvil..., Parte I
Broadcast Receivers con GPS, o cómo saber si tu novia dice la verdad cuando se va con sus amigas..., Parte II
Solo funcionará si la supuesta novia ha sido tan "inocente" como para dejar que le instales aplicaciones sin revisar nunca los permisos que necesitan y luego dejar que las ejecutes al menos una vez! Además como es código antiguo no hace el tratamiento debido de los permisos.
Y por último (creo) podría ser delito penado por la ley!