

CAPÍTULO 6.

Servicios, notificaciones y receptores de anuncios

Las aplicaciones que hemos creado hasta el momento estaban formadas por una serie de actividades, cada una de las cuales permitía construir un elemento de interacción con el usuario. Una aplicación en Android dispone de otros tipos de componentes, que se estudiarán en este capítulo. Cuando necesites que parte de una aplicación se ejecute en segundo plano, debajo de otras actividades, y sin que precise de ningún tipo de interacción con el usuario, la opción más adecuada es crear un servicio. Un servicio puede estar en ejecución indefinidamente, o podemos controlarlo desde una actividad. A lo largo de este capítulo aprenderemos las facilidades proporcionadas para la creación de servicios.

Por otra parte, las notificaciones de la barra de estado constituyen un mecanismo de comunicación vital en Android. Permiten que las aplicaciones que corren en un segundo plano adviertan al usuario sobre alertas, avisos o cualquier tipo de información. Las notificaciones se representan como pequeños iconos en la barra superior de la pantalla y se utilizan habitualmente para indicar la llegada de un mensaje, una cita de calendario, una llamada perdida o cualquier otra incidencia de interés para el usuario. Se trata de una comunicación que no requiere una interacción inmediata del usuario; este puede estar utilizando otra aplicación sin ser interrumpido o puede no estar utilizando el teléfono en ese momento. Este hecho hace de las notificaciones un mecanismo de comunicación ideal para un servicio (o receptores de anuncios, como veremos a continuación). Por lo tanto, este capítulo parece el sitio ideal para describir cómo podemos crear nuestras propias notificaciones y utilizarlas desde nuestras aplicaciones.

Terminaremos el capítulo estudiando otro componente de una aplicación Android: los receptores de anuncios. Un receptor de anuncios (*Broadcast Receiver*, en inglés) permite realizar acciones cuando se producen anuncios globales de tipo *broadcast*. Existen muchos anuncios originados por el sistema (por ejemplo, *Batería baja*, *Llamada entrante*, etc.). Aunque las aplicaciones también pueden lanzar un anuncio *broadcast* o incluso crear nuevos tipos. Los receptores de anuncios te permitirán crear aplicaciones mucho más integradas en el entorno donde se ejecutan.

**Objetivos:**

- Describir el uso de servicios en Android.
- Enumerar los pasos a seguir cuando queramos crear un servicio para que una tarea se ejecute en segundo plano.
- Mostrar cómo pueden ser utilizadas las notificaciones de la barra de estado como mecanismo de comunicación eficaz con el usuario.
- Enumerar los pasos a seguir para crear un receptor de anuncios.
- Enumerar los receptores de anuncios más importantes disponibles en Android.
- Describir el uso de receptores de anuncios como mecanismo de comunicación entre aplicaciones.
- Describir el uso de un servicio como mecanismo de comunicación entre aplicaciones.

6.1. Introducción a los servicios en Android



Vídeo[tutorial]: *Los servicios en Android*



Vídeo[tutorial]: *Un servicio para ejecución en segundo plano*

En muchos casos, será necesario añadir un nuevo componente a tu aplicación para ejecutar algún tipo de acción que se ejecute en segundo plano, es decir, que no requiera una interacción directa con el usuario, pero que queramos que permanezca activo aunque el usuario cambie de actividad. Este es el momento de crear un servicio.

En Android los servicios tienen una doble función:

- La primera función permite indicar al sistema que el elemento que estamos creando ha de ejecutarse en segundo plano, normalmente durante un largo período de tiempo. Este tipo de servicios se inician mediante el método `startService()`, que indica al sistema que los ejecute de forma indefinida hasta que alguien le indique lo contrario.
- Los servicios también permiten que nuestra aplicación se comunique con otras aplicaciones, para lo cual ofreceremos ciertas funciones que podrán llamarse desde otras aplicaciones. Este tipo de servicios se

inician mediante el método `bindService()`, que permite establecer una conexión con el servicio e invocar alguno de los métodos que ofrece.

Cada vez que se crea un servicio usando `startService()` o `bindService()`, el sistema instancia el servicio y llama al método `onCreate()`. Corresponde al servicio implementar el comportamiento adecuado; habitualmente creará un hilo de ejecución (*thread*) secundario donde se realizará el trabajo.

Un servicio en sí puede ser algo muy simple. En este capítulo se verán ejemplos de servicios locales escritos en muy pocas líneas. No obstante, también pueden complicarse, como veremos al final del capítulo, cuando tratemos de invocar servicios remotos por medio de una interfaz AIDL (*Android Interface Definition Language*).

Un servicio, como el resto de los componentes de una aplicación, se ejecuta en el hilo principal del proceso de la aplicación. Por lo tanto, si el servicio necesita un uso intensivo de CPU o puede quedar bloqueado en ciertas operaciones, como el uso de redes, debes crear un hilo diferente para ejecutar estas acciones. También puedes utilizar la clase `IntentService` para lanzar un servicio en su propio hilo.

6.1.1. Ciclo de vida de un servicio

Es importante que recuerdes que un servicio tiene un ciclo de vida diferente del de una actividad. A continuación podemos ver un gráfico que ilustra su ciclo de vida:

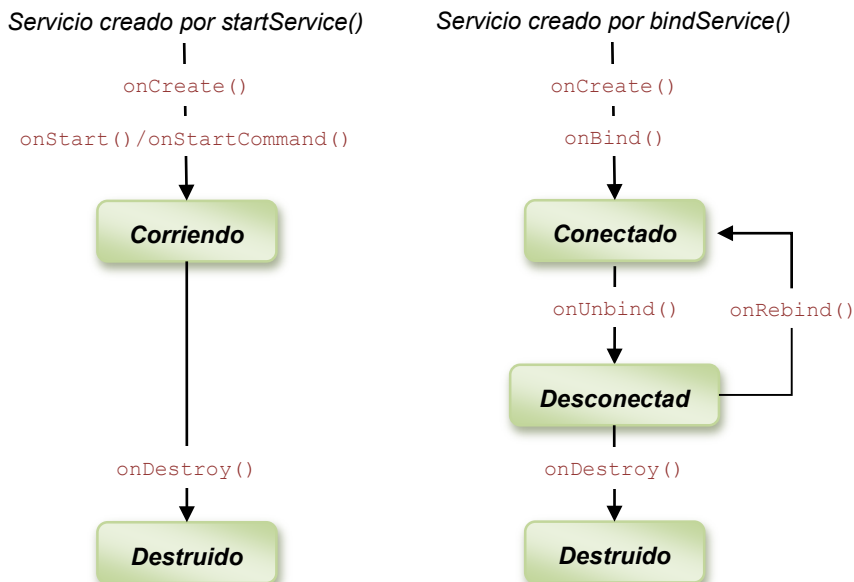


Figura 1: Ciclo de vida de los servicios.

Como acabamos de explicar, existen dos tipos de servicios en función de cómo hayan sido creados. Las funciones de estos servicios son diferentes, y por lo tanto, también su ciclo de vida.

Si el servicio se inicia mediante `startService()`, el sistema comenzará creándolo y llamando a su método `onCreate()`. A continuación llamará a su método `onStartCommand(Intent intent, int flags, int startId)`¹. El servicio continuará en ejecución hasta que sea invocado el método `stopService()` o `stopSelf()`.

***NOTA:** Si se producen varias llamadas a `startService()`, eso no supondrá la creación de varios servicios, aunque sí que se realizarán múltiples llamadas a `onStartCommand()`. No importa cuántas veces haya sido creado el servicio, parará con la primera invocación de `stopService()` o `stopSelf()`. Sin embargo, podemos utilizar el método `stopSelf(int startId)` para asegurarnos de que el servicio no parará hasta que todas las llamadas hayan sido procesadas.*

Cuando se inicia un servicio para realizar alguna tarea en segundo plano, el proceso donde se ejecuta podría ser eliminado ante una situación de baja memoria. Podemos configurar la forma en que el sistema reaccionará ante esta circunstancia según el valor que devolvamos en `onStartCommand()`. Existen dos modos principales: devolveremos `START_STICKY` si queremos que el sistema trate de crear de nuevo el servicio cuando disponga de memoria suficiente; o devolveremos `START_NOT_STICKY` si queremos que el servicio sea creado de nuevo solo cuando llegue una nueva solicitud de creación.

Conviene aclarar que en situaciones donde el sistema necesite memoria, podrá matar el proceso que contiene nuestro servicio. A la hora de elegir el proceso a eliminar, un servicio es considerado menos prioritario que las actividades visibles, aunque más prioritario que otras actividades en segundo plano. Dado que el número de actividades visibles es siempre reducido, un servicio solo será eliminado en situaciones de extrema necesidad de memoria. Por otra parte, si un cliente visible está conectado a un servicio, el servicio también será considerado como visible, siendo tan prioritario como el cliente. En el caso de un proceso que contenga varios componentes (por ejemplo, una actividad y un servicio), su prioridad se obtiene como el máximo de sus componentes.

También podemos utilizar `bindService(Intent servicio, ServiceConnection conexion, int flags)` para obtener una conexión persistente con un servicio. Si dicho servicio no está en ejecución, será creado (siempre que el `flag BIND_AUTO_CREATE` esté activo), llamándose al método `onCreate()`, pero no se llamará a `onStartCommand()`. En su lugar se llamará al método `onBind(Intent intencion)`, que ha de devolver al cliente un objeto `IBinder` a través del cual se podrá establecer una comunicación entre cliente y servicio. Esta comunicación se establece por medio de una interfaz escrita en AIDL, que permite el intercambio de objetos entre aplicaciones que corren en

¹ En versiones de la API inferiores a 2.0, el método llamado será `onStart()`. En versiones recientes se mantiene por razones de compatibilidad.

procesos separados. El servicio permanecerá en ejecución tanto tiempo como la conexión esté establecida, independientemente de que se mantenga o no la referencia al objeto `IBinder`.

También es posible diseñar un servicio que pueda ser arrancado de ambas formas (`startService()` y `bindService()`). Este servicio permanecerá activo si ha sido creado desde la aplicación que lo contiene o si recibe conexiones desde otras aplicaciones.

6.1.2. Permisos

Podemos conseguir acceso global a un servicio declarado en la etiqueta `<service>` de `AndroidManifest.xml`. También podemos definir un permiso para restringir su acceso. En este caso, las aplicaciones han de declarar este permiso, con el correspondiente `<uses-permission>` en su propio manifiesto.

Podemos definir un permiso para arrancar, parar o conectarse a un servicio. De forma adicional, podemos restringir el acceso a funciones específicas de las ofertadas por un servicio. Para este propósito, podemos llamar al principio de nuestra función a `checkCallingPermission(String)` para verificar si el cliente dispone de un permiso en concreto. Para más información sobre permisos, se recomienda la lectura del capítulo 7.

6.2. Un servicio para ejecución en segundo plano

Dentro de los dos usos de un servicio, el más frecuente es permitirnos ejecutar parte de nuestra aplicación en segundo plano.



Ejercicio: *Un servicio para ejecución en segundo plano de reproducción de música*

Veamos un ejemplo de servicio que corre en el mismo proceso de la aplicación que lo utiliza. El servicio será creado con la finalidad de reproducir una música de fondo y podrá ser arrancado y detenido desde la actividad principal.

1. Crea un nuevo proyecto con los siguientes datos:

Application Name: Servicio Música

☒ Phone and Tablet

Minimum SDK: API 16 Android 4.1 (Jelly Bean)

Add an activity: Empty Activity

2. Reemplaza el código del *layout* `activity_main.xml` por:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
```

```

    android:layout_height="match_parent">
    <TextView android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Servicio de reproducción de música"/>
    <Button android:id="@+id/boton_arrancar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Arrancar servicio"/>
    <Button android:id="@+id/boton_detener"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Detener servicio"/>
</LinearLayout>

```

Se trata de un *layout* muy sencillo, con un texto y dos botones:

Servicio de reproducción de música

Arrancar servicio

Detener servicio

3. Reemplaza el código de la actividad por:

```

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

        Button arrancar = findViewById(R.id.boton_arrancar);
        arrancar.setOnClickListener(new OnClickListener() {
            public void onClick(View view) {
                startService(new Intent(MainActivity.this,
                    ServicioMusica.class));
            }
        });
        Button detener = findViewById(R.id.boton_detener);
        detener.setOnClickListener(new OnClickListener() {
            public void onClick(View view) {
                stopService(new Intent(MainActivity.this,
                    ServicioMusica.class));
            }
        });
    }
}

```

4. Crea la nueva clase, *ServicioMusica*, con el siguiente código:

```

public class ServicioMusica extends Service {
    MediaPlayer reproductor;
}

```

```

@Override public void onCreate() {
    Toast.makeText(this, "Servicio creado",
        Toast.LENGTH_SHORT).show();
    reproductor = MediaPlayer.create(this, R.raw.audio);
}

@Override
public int onStartCommand(Intent intenc, int flags, int idArranque) {
    Toast.makeText(this, "Servicio arrancado "+ idArranque,
        Toast.LENGTH_SHORT).show();

    reproductor.start();
    return START_STICKY;
}

@Override public void onDestroy() {
    Toast.makeText(this, "Servicio detenido",
        Toast.LENGTH_SHORT).show();

    reproductor.stop();
    reproductor.release();
}

@Override public IBinder onBind(Intent intencion) {
    return null;
}
}

```

5. Edita el fichero `AndroidManifest.xml` y añade la siguiente línea dentro de la etiqueta `<application>`:

```
<service android:name=".ServicioMusica" />
```

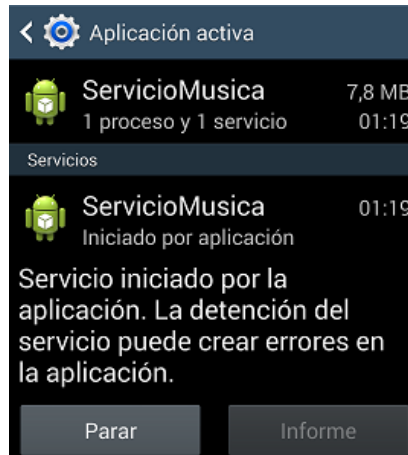
6. Crea una nueva carpeta que se llame `raw` dentro de la carpeta `res`. Arrastra a su interior el fichero `audio.mp3`.

NOTA: Puedes usar cualquier fichero de música compatible, siempre que el nombre sea audio. Ej. <http://www.dcomg.upv.es/~jtomas/android/ficheros/audio.mid>

7. Ejecuta la aplicación y comprueba su funcionamiento. Puedes terminar la actividad pulsando la tecla "retorno" y verificar que el servicio continúa en marcha.
8. Verifica que, aunque pulses varias veces el botón *Arrancar servicio*, este no vuelve a crearse, pero sí que vuelve a llamarse al método `onStartCommand()`. Además, con solo una vez que pulses en *Detener servicio*, este parará.

Ejecuta la aplicación y pon en funcionamiento el servicio. En un dispositivo con versión 6.0 o superior, asegúrate que estén activadas las opciones de desarrollador. Accede a *Ajustes > Opciones del desarrollador > Servicios en ejecución*. En una versión anterior a la 6.0 accede a *Ajustes > Más > Administrador de aplicaciones*. Selecciona la pestaña *EN EJECUCIÓN* y busca *ServicioMusica*. Desde aquí puedes obtener información y detener el servicio.

NOTA: En versiones muy antiguas no está disponible esta información.



6.2.1. El método `onStartCommand()`

El método `onStartCommand()` aparece a partir del nivel de API 5, en sustitución de `onStart()`. Se llama cada vez que un cliente inicializa un servicio mediante el método `startService()`. Veamos con más detalle cómo pueden ser utilizados sus parámetros para obtener información valiosa:

```
public int onStartCommand(Intent intencion, int flags, int idArranque)
```

Los parámetros se detallan a continuación:

`intencion` Un objeto `Intent` que se indicó en la llamada `startService(Intent)`.

`flags` Información adicional sobre cómo arrancar el servicio. Puede ser 0, `START_FLAG_REDELIVERY` o `START_FLAG_RETRY`. Un valor distinto de 0 se utiliza para reiniciar un servicio tras detectar algún problema.

`idArranque` Un entero único que representa la solicitud de arranque específica. Usar este mismo entero en el método `stopSelfResult(int idArranque)`.

`retorna` Describe cómo ha de comportarse el sistema cuando el proceso del servicio sea matado una vez que el servicio ya se ha inicializado. Esto puede ocurrir en situaciones de baja memoria. Los siguientes valores están permitidos:

`START_STICKY`: Cuando sea posible, el sistema tratará de recrear el servicio. Se realizará una llamada a `onStartCommand()`, pero con el parámetro `intencion` igual a `null`. Esto tiene sentido cuando el servicio puede arrancar sin información adicional como, por ejemplo, el servicio mostrado para la reproducción de música de fondo.

`START_NOT_STICKY`: El sistema no tratará de volver a crear el servicio; por lo tanto, el parámetro `intencion` nunca podrá ser igual a `null`.

Esto tiene sentido cuando el servicio no puede reanudarse una vez interrumpido.

`START_REDELIVER_INTENT`: El sistema tratará de volver a crear el servicio. El parámetro `intencion` será el que se utilizó en la última llamada `startService(Intent)`.

`START_STICKY_COMPATIBILITY`: Versión compatible de `START_STICKY`, que no garantiza que `onStartCommand()` sea llamado después de que el proceso sea matado.



Preguntas de repaso: Servicios

6.3. Un servicio en un nuevo hilo con IntentService

A la hora de diseñar aplicaciones en Android hay que tener muy en cuenta que todos los componentes (actividades, servicios y receptores de anuncios) se van a ejecutar en el hilo principal de la aplicación. Dado que este hilo ha de estar siempre disponible para atender a los eventos generados por el usuario, nunca debe ser bloqueado. Es decir, cualquier proceso que requiera un tiempo importante no ha de ser ejecutado desde este hilo. En su lugar hay que crear un nuevo hilo para que realice este proceso y así dejar libre al hilo principal para que este pueda seguir procesando nuevos eventos.

Podemos crear un nuevo hilo utilizando la clase estándar de Java `Thread`, tal y como se ha explicado en el capítulo 5. Para automatizar este proceso, Android nos proporciona la clase `AsyncTask`. También nos proporciona la clase `IntentService`, cuando queramos lanzar un servicio en un nuevo hilo. En este apartado veremos qué ocurre cuando un servicio bloquea el hilo principal y cómo solucionarlo mediante la clase `IntentService`.



Ejercicio: Un servicio que bloquea el hilo principal

Muchos servicios han de realizar costosas operaciones o han de esperar a que concluyan lentas operaciones en la red. En ambos casos hay que tener la precaución de no bloquear el hilo principal. De hacerlo, el resultado puede ser catastrófico, como se muestra en este ejercicio.

1. Crea un nuevo proyecto que se llame `IntentService` y cuyo nombre de paquete sea `com.example.intentservice`.
2. Reemplaza el código del `layout` principal por:

```
<LinearLayout 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:orientation="vertical" >
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
    <EditText
        android:id="@+id/entrada"
        android:layout_width="0dip"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:inputType="numberDecimal"
        android:text="2.2" >
        <requestFocus />
    </EditText>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="calcularOperacion"
        android:text="Calcular operación" />
</LinearLayout>
<TextView
    android:id="@+id/salida"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:text=" "
    android:textAppearance="?android:attr/textAppearanceMedium"/>
</LinearLayout>

```

3. Reemplaza el código de `MainActivity` por el siguiente:

```

public class MainActivity extends Activity {
    private EditText entrada;
    public static TextView salida;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        entrada = findViewById(R.id.entrada);
        salida = findViewById(R.id.salida);
    }

    public void calcularOperacion(View view) {
        double n = Double.parseDouble(entrada.getText().toString());
        salida.append(n + "^2 = ");
        Intent i = new Intent(this, ServicioOperacion.class);
        i.putExtra("numero", n);
        startService(i);
    }
}

```

Observa que la variable `salida` ha sido declarada como **public static**. Esto nos permitirá acceder a esta variable desde otras clases. Se llamará al método `calcularOperacion()` cuando se pulse el botón. Comienza obteniendo el

valor real introducido en *entrada*. Se muestra la operación a realizar por *salida*. Luego, se crea una nueva intención con nuestro contexto y la clase con el servicio que se define a continuación. Luego se le añade un extra con el valor introducido. Finalmente se arranca el servicio.

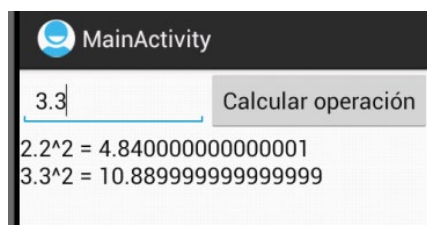
4. Crea la clase *ServicioOperacion* con el siguiente código:

```
public class ServicioOperacion extends Service {
    @Override
    public int onStartCommand(Intent i, int flags, int idArranque){
        double n = i.getExtras().getDouble("numero");
        SystemClock.sleep(5000);
        MainActivity.salida.append(n*n + "\n");
        return START_NOT_STICKY;
    }

    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }
}
```

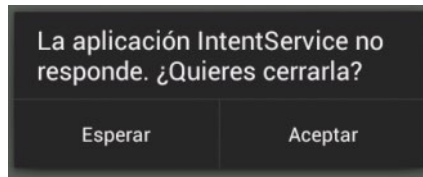
Cuando se arranca el servicio se llamará al método *onStartCommand()*. Este comienza obteniendo el valor a calcular a partir de un extra. Luego vamos a simular que se realizan un gran número de operaciones. Para ello vamos a bloquear el hilo durante 5000 ms (5 segundos) utilizando el método *sleep()*. Una vez terminado, el resultado se muestra directamente en el *TextView salida*. Esta forma de trabajar no resulta muy recomendable. Se ha realizado así para ilustrar como es posible acceder al sistema gráfico de Android dado que estamos en el hilo principal. Finalmente, devolvemos *START_NOT_STICKY* para indicar al sistema que si por fuerza mayor ha de destruir el servicio, no hace falta que lo vuelva a crear.

5. Recuerda registrar el servicio en *AndroidManifest.xml*.
6. Ejecuta la aplicación. El resultado ha de ser similar al siguiente:



Observa como mientras se realiza la operación el usuario no puede pulsar el botón ni modificar el EditText. El usuario tendrá la sensación de que la aplicación está bloqueada.

7. Modifica el tiempo de retardo para que este sea de 25 seg. (*sleep(25000)*). Ejecuta de nuevo la aplicación y observa como el sistema nos mostrará el siguiente error:



8. Para que esto no bloquee el hilo principal podemos utilizar un `IntentService`. En el siguiente ejercicio mostraremos cómo realizarlo.

6.3.1. La clase `IntentService`

Utilizaremos la clase `IntentService` en lugar de `Service` cuando queramos un servicio que se ejecute en su propio hilo. Esta clase tiene un constructor donde hay que indicar en un `String` el nombre que queremos dar al servicio. Lo habitual será que cuando extendamos esta clase en el constructor llamemos al constructor padre pasándole este nombre. A continuación se muestra un ejemplo de código:

```
public class MiServicio extends IntentService{

    public MiServicio () {
        super("Nombre de mi servicio");
    }

    @Override
    protected void onHandleIntent(Intent intencion) {
        ...
    }
}
```

El siguiente método que hay que sobrescribir es `onHandleIntent`. Este método se lanzará cada vez que se arranque el servicio, pero en este caso se lanzará en hilo nuevo. A través del parámetro *intención* se podrán enviar datos en forma de extras. Es importante destacar que si se lanzan varias peticiones de servicio, estas se pondrán en una cola. Se irán atendiendo una tras otra sin que haya dos a la vez en ejecución. Este comportamiento puede ser interesante para algunas tareas, pero no para otras. Por ejemplo, si tenemos que implementar un servicio de descarga de ficheros, seguramente será más interesante permitir la descarga de varios ficheros en paralelo y no tener que descargarlos de uno en uno.

Finalmente, para lanzar un `IntentService` hay que usar `startService()`. Se realiza exactamente igual que para lanzar un `Service`.



Ejercicio: *Un servicio en su propio hilo*

En este ejercicio aprenderemos a crear servicios que se ejecutan en un hilo de ejecución diferente del principal utilizando la clase `IntentService`. Además veremos

algunas limitaciones de este tipo de servicios, como la imposibilidad de acceder al sistema gráfico.

1. Abre el proyecto `IntentService` creado en el ejercicio anterior.
2. Crea la clase `IntentServiceOperacion` con el siguiente código:

```
public class IntentServiceOperacion extends IntentService{
    public IntentServiceOperacion() {
        super("IntentServiceOperacion");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        double n = intent.getExtras().getDouble("numero");
        SystemClock.sleep(5000);
        MainActivity.salida.append(n*n + "\n");
    }
}
```

3. En `MainActivity` reemplaza la línea:

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

por:

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

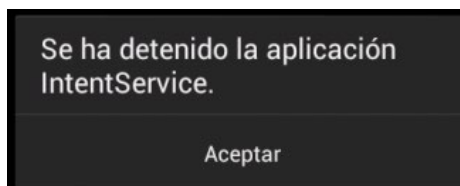
4. En `AndroidManifest.xml` reemplaza la línea:

```
<service android:name=".ServicioOperacion" />
```

por:

```
<service android:name=".IntentServiceOperacion" />
```

5. Ejecuta la aplicación. Tras pulsar el botón el resultado ha de ser:



6. Abre la vista `LogCat` y busca el siguiente Error:

```
FATAL EXCEPTION: IntentService[IntentServiceOperacion]
android.view.ViewRootImpl$CalledFromWrongThreadException: Only the
original thread that created a view hierarchy can touch its views.
WS.
```

Te indica que solo desde el hilo principal se va a poder interactuar con las vistas de la interfaz de usuario. También está prohibido usar la clase `Toast` desde otros hilos.

Como un hilo que hemos creado pertenece al mismo proceso que el hilo principal, compartimos con este todas las variables. Para devolver el valor calculado, podríamos implementar un método o variable públicos, tanto en la clase del servicio como de la actividad. No obstante, vamos a resolver este problema utilizando un mecanismo más elegante, los receptores de anuncios. Se explica en el siguiente apartado.



Preguntas de repaso: *Servicios e hilos*

6.4. Las notificaciones de la barra de estado

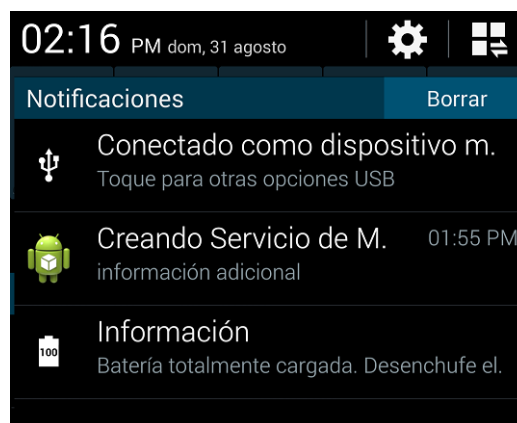


Vídeo[tutorial]: *Notificaciones en Android*

La barra de estado de Android se encuentra situada en la parte superior de la pantalla. La parte izquierda de esta barra está reservada para visualizar notificaciones. Cuando se crea una nueva notificación, aparece un pequeño icono que permanecerá en la barra para recordar al usuario la notificación.



El usuario puede arrastrar la barra de notificaciones hacia abajo, para mostrar la lista de las notificaciones por leer. A continuación se muestra un ejemplo:

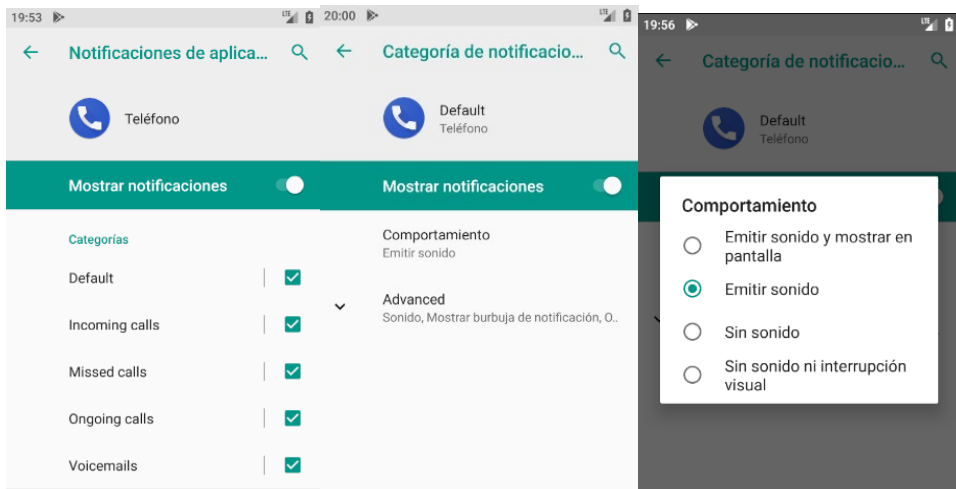


Una notificación puede ser creada por un servicio o por una actividad. Aunque dado que la actividad dispone de su propia interfaz de usuario, parece que las notificaciones son el mecanismo de interacción más interesante del que disponen

los servicios. Las notificaciones pueden crearse desde un segundo plano, sin interferir con la actividad que en ese momento esté utilizando el usuario.

Para lanzar una notificación desde Android 8.0 Oreo (API 26) es necesario hacerlo dentro de un canal de notificación. Si se te olvida crear el canal, la notificación no aparecerá. Esto permite a los usuarios tratar todas las notificaciones de un mismo canal de forma conjunta (desactivarla, configurar sonido...). Una misma aplicación puede crear varios canales, uno por cada tipo de notificación. Por ejemplo, en WhatsApp podríamos crear un canal para cada uno de los grupos.

Para comprender mejor este concepto, en un terminal (o emulador) con versión 8 o superior, accede a Ajustes / Aplicaciones y notificaciones / Teléfono / Notificaciones de la aplicación. Se mostrarán todos los canales de notificación creados por esta aplicación. Observa como en *Ajustes* a los *canales* se les llama *categorías*:



Desde esta pantalla podrás desactivar cualquier canal de notificaciones. Si pulsas sobre su nombre, podrás configurar el canal. En *Comportamiento* puedes configurar la importancia de una notificación para determinar si el usuario ha de ser interrumpido, tanto visual como acústicamente. Los cuatro posibles niveles se muestran a la derecha. También podemos configurar el sonido asociado o si queremos que aparezcan en modo no molestar.



Ejercicio: Creación de una notificación

1. Abre el proyecto ServicioMusica.
2. Asegúrate de que tu proyecto incluye la librería de compatibilidad v7 o superior. Para ello, en *Project/Gradle Script/build.gradle (Module: app)* añade la línea:

```
dependencies {
    ...
    implementation 'com.android.support:appcompat-v7:27.1.1'
}
```

3. Declara la siguiente variables al comienzo de la clase `ServicioMusica`:

```
private NotificationManager notificationManager;
static final String CANAL_ID = "mi_canal";
static final int NOTIFICACION_ID = 1;
```

4. Para crear una nueva notificación añade al principio del método `onStartCommand()` las siguientes líneas:

```
notificationManager = getSystemService(NOTIFICATION_SERVICE);
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    NotificationChannel notificationChannel = new NotificationChannel(
        CANAL_ID, "Mis Notificaciones",
        NotificationManager.IMPORTANCE_DEFAULT);
    notificationChannel.setDescription("Descripción del canal");
    notificationManager.createNotificationChannel(notificationChannel);
}
NotificationCompat.Builder notificacion =
    new NotificationCompat.Builder(MainActivity.this, CANAL_ID)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle("Título")
        .setContextText("Texto de la notificación.");
notificationManager.notify(NOTIFICACION_ID, notificacion.build());
```

Lo primero que necesitamos es una referencia al `NotificationManager` que permite manejar las notificaciones del sistema.

El siguiente paso es crear el canal. Solo puede hacerse en dispositivos con versión 8 o superior; no obstante, es obligatorio que crees el canal, de lo contrario, las notificaciones no se crearán en dispositivos con estas versiones. Para crear el canal has de indicar un ID, un nombre y un nivel de importancia. El nivel de importancia corresponde con los cuatro niveles de comportamiento mostrados en las capturas de *Ajustes*.

La notificación se crea utilizando una clase especial `Builder` que dispone de varios métodos `set` para configurarla. `setContentTitle()` permite indicar el título que describe la notificación y `setContextText()`, información más detallada. Con `setSmallIcon()` indicamos el icono a visualizar. En el ejemplo usamos el mismo que el de la aplicación, aunque no resulta muy adecuado dado que estos iconos han de seguir una estética concreta.

El método `notify()` es el encargado de lanzar la notificación. En el primer parámetro se indica un id para poder identificar esta notificación en un futuro y en el segundo la notificación.

5. Ejecuta la aplicación y verifica el resultado.



Ejercicio: Lanzar una actividad desde una notificación

Cuando arrastras la barra de notificaciones hacia abajo, para mostrar la lista de notificaciones, y pulsas sobre una de ellas, es habitual que se abra una

actividad para realizar acciones relacionadas con la notificación. En este ejercicio aprenderemos a asociar una actividad a una notificación.

1. Tras la creación de la variable `notificacion` introducida en el ejercicio anterior, añade el siguiente código:

```
PendingIntent intencionPendiente = PendingIntent.getActivity(  
    this, 0, new Intent(this, MainActivity.class), 0);  
notificacion.setContentIntent(intencionPendiente);
```

Este código asocia una actividad que se ejecutará cuando el usuario pulse sobre la notificación. Para ello, se crea un `PendingIntent` (intención pendiente que se ejecutará más adelante) asociado a la actividad `MainActivity`. Por supuesto, también puedes crear una nueva actividad para usarla exclusivamente con este fin. En un ejemplo más complejo, puedes pasar los parámetros adecuados a través del `Intent`, para que la actividad conozca los detalles específicos que provocaron la notificación (por ejemplo, el número de teléfono que provocó la llamada perdida).

2. Ejecuta la aplicación y verifica el resultado.



Ejercicio: Eliminar una notificación

Eliminar una notificación desde código resulta muy sencillo. Se describe en este ejercicio:

1. Queremos que si el servicio deja de estar activo, elimine la notificación. Para ello añade en `onDestroy()`:

```
notificationManager.cancel(NOTIFICACION_ID);
```

Este paso es opcional. Muchas notificaciones han de permanecer visibles aunque el servicio que las creó sea destruido. En nuestro caso, dado que estamos anunciando que un servicio de reproducción de música está activado, la notificación deja de tener sentido al desaparecer el servicio.

2. Ejecuta la aplicación y verifica que al parar el servicio la notificación desaparece.



Práctica: Uso del servicio de música en Asteroides

1. Copia la clase `ServicioMusica` del ejercicio anterior en el proyecto Asteroides.
2. Corrige los errores que hayan aparecido para adaptarla al nuevo proyecto.
3. En el ejercicio anterior, cuando se visualizaban los detalles de la notificación se podía lanzar la actividad `MainActivity` del proyecto `ServicioMusica`. Ahora ha de lanzarse la actividad `MainActivity` del proyecto `Asteroides`.

- Si realizas el punto anterior simplemente lanzando la actividad `MainActivity`, cuando el usuario pulse sobre la notificación el sistema lanzará una nueva tarea, aunque ya exista una previa. Si te interesa que no se lance una nueva tarea cuando ya exista una previa, añade la línea en negrita en `AndroidManifest.xml`.

```
<activity android:name=".MainActivity"
    android:label="@string/app_name"
    android:launchMode="singleTask">
```

- Lanza el servicio en el método `onCreate()` de la actividad `MainActivity`. Para el servicio en el método `onDestroy()`.



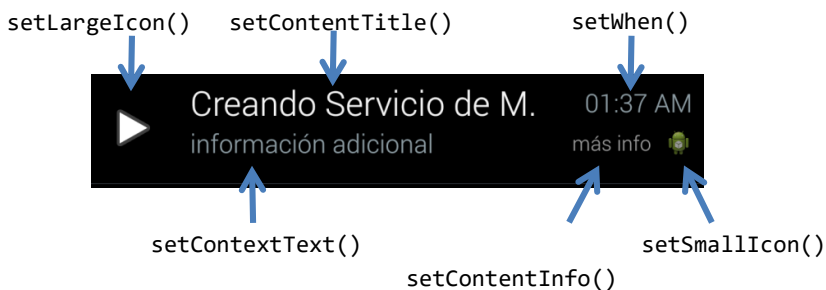
Ejercicio: Características avanzadas con notificaciones

Lo aprendido hasta ahora es suficiente para cubrir la mayoría de las notificaciones. No obstante, en este ejercicio aprenderemos a configurar otras características que en ciertos casos podrían ser interesantes.

- En el proyecto `ServicioMusica` añade las siguientes líneas en la creación del objeto `NotificationCompat.Builder`:

```
.setLargeIcon(BitmapFactory.decodeResource(getResources(),
    android.R.drawable.ic_media_play))
.setWhen(System.currentTimeMillis() + 1000 * 60 * 60)
.setContentInfo("más info")
.setTicker("Texto en barra de estado")
```

El siguiente esquema muestra dónde se visualiza parte de la información que acabamos de introducir:



El método `setLargeIcon()` permite visualizar un icono de mayor resolución que se muestra a la izquierda de la notificación. `setWhen()` permite indicar la hora en que ocurrió el evento. Solo actúa a efectos de visualización. Aunque se indique una hora futura, la notificación se lanzará inmediatamente. Las notificaciones en el panel se ordenan por este tiempo. `setContentInfo()` muestra un texto ubicado en la parte inferior derecha de la notificación. Finalmente `setTicker()` muestra un texto en la barra de estado, durante unos segundos, cuando la notificación se crea por primera vez.

2. Ejecuta la aplicación y verifica el resultado.

6.4.1. Configurando tipos de avisos en las notificaciones

Como hemos comentado, una notificación puede utilizar diferentes métodos para alertar al usuario de que se ha producido. Veamos algunas opciones.

Asociar un sonido

Si consideras que una notificación es muy urgente y deseas que el usuario pueda conocerla de forma inmediata, puedes asociarle un sonido, que se reproducirá cuando se produzca la notificación.

El usuario puede definir un sonido por defecto para las notificaciones. Si quieres asociar el sonido de notificaciones por defecto, utiliza la siguiente sentencia en la creación del objeto `NotificationCompat.Builder`:

```
.setDefaults(Notification.DEFAULT_SOUND)
```

Si prefieres reproducir un sonido personalizado para la notificación, puedes copiar un fichero de audio en la carpeta `res/raw` del proyecto (por ejemplo, el fichero `explosión.mp3`) y añadir la siguiente sentencia:

```
.setSound(Uri.parse("android.resource://" + getPackageName() + "/" + R.raw.explosion));
```

Añadiendo vibración

También es posible alertar al usuario haciendo vibrar el teléfono. Puedes utilizar la vibración por defecto:

```
.setDefaults(Notification.DEFAULT_VIBRATE);
```

O, por el contrario, tu propio patrón de vibración:

```
.setVibrate(new long[] { 0,100,200,300 })
```

El *array* define un patrón de longitudes expresadas en milisegundos, donde el primer valor es el tiempo sin vibrar; el segundo es el tiempo vibrado; el tercero, el tiempo sin vibrar, y así sucesivamente. Este *array* puede ser tan largo como queramos, pero solo se activará una vez, no se repetirá de forma cíclica.

Añadiendo parpadeo de LED

Algunos móviles disponen de diodos LED que pueden utilizarse para avisar al usuario de que se ha producido una notificación. Este método es muy interesante si el grado de urgencia del aviso no es lo suficientemente alto para usar uno de los métodos anteriores. Podemos utilizar el aviso de LED configurado por defecto:

```
.setDefaults(Notification.DEFAULT_LIGHTS)
```

O podemos definir una cadencia de tiempo y color específica para nuestra notificación:

```
.setLights(Color.RED, 3000, 1000);
```

En el ejemplo anterior indicamos que queremos que el LED se ilumine en color rojo durante 3000 ms y luego esté apagado durante 1000 ms. Esta secuencia se repetirá de forma cíclica hasta que el usuario atienda la notificación.

Conviene destacar que no todos los móviles disponen de un LED para este propósito, y algunos dispositivos con LED no soportan notificaciones con parpadeo. Además, no todos los colores pueden ser utilizados. Otro aspecto a tener en cuenta es que el sistema solo activa el LED cuando la pantalla está apagada.



Práctica: Una notificación de socorro

1. En el proyecto anterior, crea un nuevo botón.
2. Al pulsar este botón se lanzará una nueva notificación que mostrará el texto «¡SOCORRO!».
3. El audio de la notificación será una grabación de voz que diga «¡SOCORRO!».
4. La notificación hará vibrar el teléfono con el mensaje internacional de socorro S.O.S. codificado en Morse. Para ello, haz vibrar el teléfono con una sucesión de tres pulsaciones cortas, tres largas y otras tres cortas (. . . - - - . . .).

Desde Android 8 resulta más interesante configurar estas características especiales en un canal de notificación, en lugar de hacerlo en cada notificación. Para ello no tienes más que añadir las líneas subrayadas, cuando crees el canal:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    NotificationChannel notificationChannel = new NotificationChannel(
        CANAL_ID, "Mis Notificaciones",
        NotificationManager.IMPORTANCE_DEFAULT);
    notificationChannel.setDescription("Descripcion del canal");
    notificationChannel.enableLights(true);
    notificationChannel.setLightColor(Color.RED);
    notificationChannel.setVibrationPattern(new long[]{0, 100, 300, 100});
    notificationChannel.enableVibration(true);
    notificationManager.createNotificationChannel(notificationChannel);
}
```



Preguntas de repaso: Notificaciones

6.4.2. Servicios en primer plano

Hasta Android 8 una aplicación podría ejecutarse libremente en segundo plano. Sin embargo, a partir de esta versión se va a limitar la ejecución en segundo plano.

Los motivos que ha llevado a Google a tomar esta decisión es que las aplicaciones en segundo plano son grandes consumidoras de batería. Además, afectan al rendimiento de la aplicación en primer plano, lo que supone una peor experiencia de usuario. (especialmente por el consumo de RAM).

Estas restricciones no se aplican si la aplicación se encuentra en primer plano. Se considera que una aplicación se encuentra en primer plano si se cumple alguno de los siguientes puntos:

- Tiene una actividad visible, independientemente de que la actividad se haya iniciado o esté en pausa.
- Otra aplicación en primer plano está conectada a la aplicación, ya sea por vinculación a uno de sus servicios o por el uso de uno de sus proveedores de contenido.
- Tiene un servicio en primer plano.

En concreto las restricciones a una aplicación que no está en primer plano es:

- No podemos arrancar un servicio.
- Cuando la aplicación pase a segundo plano, los servicios creados permanecen activos varios minutos. Pasado este tiempo son detenidos.

Un servicio en primer plano es un servicio que se considera que el usuario es consciente de su actividad. Para conseguir esto, el servicio en primer plano debe proporcionar una notificación para la barra de estado.

Esta notificación es de tipo “en curso”, lo que significa que la notificación no se puede descartar, salvo que el servicio se detenga o se quite del primer plano.

Por ejemplo, un reproductor de música que reproduce música desde un servicio se debe configurar para que se ejecute en primer plano, porque el usuario está explícitamente al tanto de su operación. La notificación en la barra de estado puede indicar la canción actual y permitir que el usuario lance una actividad para interactuar con el reproductor de música.

Para solicitar que tu servicio se pase a primer plano, créalo como se ha indicado y llama al siguiente método:

```
startForeground(NOTIFICACION_ID, notificacion);
```

Los dos parámetros son los mismos que usábamos para lanzar una notificación:

```
notificationManager.notify(NOTIFICACION_ID, notificacion);
```

Pero cuidado, no has de lanzar la notificación, solo has de prepararla y pasarla en el parámetro.

Para quitar un servicio de primer plano utiliza el siguiente método:

```
stopForeground(quitarNotificacion);
```

El parámetro es un valor booleano, donde indicamos si queremos que la notificación sea retirada.

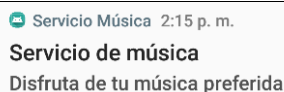


Ejercicio: Servicio de música en primer plano

1. Abre el proyecto Servicio de música y ejecútalo en un dispositivo con una versión de Android 8.0, o superior.
2. Sin salir de la actividad principal, pon otra aplicación en primer plano. La música seguirá escuchándose. Sin embargo, pasado unos 40 segundos, el servicio será detenido, llamando al método `onDestroy()`.
3. Para evitarlo, tenemos que hacer que el servicio sea considerado de primer plano asociándole una notificación. Dentro de `onStartCommand()` crea una notificación tal y como se hizo en el ejercicio [Creación de una notificación](#). Personaliza los textos para que sean adecuados a un servicio de música.
4. Queremos que si el usuario pulsa sobre la notificación se abra `MainActivity`. Para ello sigue las instrucciones del ejercicio [Lanzar una actividad desde una notificación](#).
5. Elimina la línea donde se lanzaba la notificación y reemplazala por una llamada a `startForeground()`:

```
notificationManager.notify(NOTIFICACION_ID, notificacion.build());  
startForeground(NOTIFICACION_ID, notificacion.build());
```

6. Ejecuta la aplicación. Comprueba como aparece una notificación que indica que el servicio está en marcha.
7. Si muestras los detalles de las notificaciones y tratas de descartarla, verás que no es posible.



8. Pasa la aplicación a segundo plano. Verifica que el servicio no es detenido.
9. Pulsa sobre la notificación, se abrirá la actividad con los dos botones. Desde aquí sí que podrás detener el servicio.

6.5. Receptores de anuncios

Un receptor de anuncios (`BroadcastReceiver`) recibe anuncios globales de tipo *broadcast* y reacciona ante ellos. Existen muchos originados por el sistema **como**, por ejemplo, *Batería baja* o *Llamada entrante* (más adelante se muestra una

tabla). Aunque las aplicaciones también pueden lanzar sus propios anuncios *broadcast*. Un anuncio *broadcast* se envía y se recibe en forma de intención, donde se describe el evento o la acción que ha ocurrido y se puede adjuntar información adicional.

Los receptores de anuncios no tienen interfaz de usuario, aunque pueden iniciar una actividad o crear una notificación para informar al usuario. El ciclo de vida de un `BroadcastReceiver` es muy sencillo, solo dispone del método `onReceive()`. De hecho, un objeto `BroadcastReceiver` solo existe durante la llamada a `onReceive()`. El sistema crea el `BroadcastReceiver`, llama a este método y cuando termina destruye el objeto.

El método `onReceive()` es ejecutado por el hilo principal de la aplicación. Por lo tanto, no debe bloquear el sistema (véase el ciclo de vida de una actividad). Si tienes que realizar una acción que puede bloquear el sistema, tendrás que lanzar un hilo secundario. Si queremos una acción persistente en el tiempo, resulta muy frecuente lanzar un servicio. Desde un `BroadcastReceiver` no se puede mostrar un cuadro de diálogo o unirse a un servicio (`bindService()`). Para lo primero, en su lugar puedes lanzar una notificación. Para lo segundo, puedes utilizar `startService()` para arrancar un servicio.

Una aplicación puede registrar un receptor de anuncios de dos maneras: en *AndroidManifest.xml* y en tiempo de ejecución mediante el método `registerReceiver()`.

6.5.1. Receptor de anuncios registrado en *AndroidManifest.xml*

Registrar un receptor de anuncios desde *AndroidManifest.xml* es muy sencillo. No tienes más que introducir las siguientes líneas en *AndroidManifest.xml* dentro de la etiqueta `<application>`:

```
<receiver android:name=".ReceptorAnuncio" >
    <intent-filter>
        <action android:name="android.intent.action.BATTERY_LOW" />
    </intent-filter>
</receiver>
```

En segundo lugar tienes que crear la clase `ReceptorAnuncio`. Se llamará al método `onReceive()` cuando el sistema lance el anuncio *broadcast* `BATTERY_LOW`. Esto ocurrirá cuando detecte un nivel bajo de batería.

```
public class ReceptorAnuncio extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //...
    }
}
```

La principal ventaja de declarar un receptor de anuncios en el manifiesto es que en caso de que nuestra aplicación no esté en ejecución, esta será puesta en ejecución de forma automática.

A partir de Android 8 no se pueden registrar receptores de anuncios desde el Manifiesto, a no ser que se encuentren en la lista blanca de anuncios permitidos². Si no están en esta lista, podremos programar nuestro receptor de anuncio por código.



Ejercicio: Un receptor de anuncios

Primero has de realizar el ejercicio: “Las notificaciones de la barra de estado”.

1. Crea un nuevo proyecto y llámalo LlamadaEntrante.
2. Edita *AndroidManifest.xml* y añade dentro de la etiqueta `<application>` las siguientes líneas:

```
<receiver android:name="ReceptorLlamadas" >
  <intent-filter >
    <action android:name="android.intent.action.PHONE_STATE"/>
  </intent-filter>
</receiver>
```

De esta forma registramos un receptor de anuncios que se activará cuando se produzca una llamada.

3. Tenemos que pedir permiso para leer el estado del teléfono. Añade la siguiente línea dentro de `<manifest>`.

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

4. Crea una nueva clase que se llame `ReceptorLlamadas` con el siguiente código:

```
public class ReceptorLlamadas extends BroadcastReceiver {

    @Override public void onReceive(Context context, Intent intent) {
        // Sacamos información de la intención
        String estado = "", numero = "";
        Bundle extras = intent.getExtras();
        if (extras != null) {
            estado = extras.getString(TelephonyManager.EXTRA_STATE);
            if (estado.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
                numero = extras.getString(
                    TelephonyManager.EXTRA_INCOMING_NUMBER);
                String info = estado + " " + numero;
                Log.d("ReceptorAnuncio", info + " intent=" + intent);

                // Creamos Notificación
            }
        }
    }
}
```

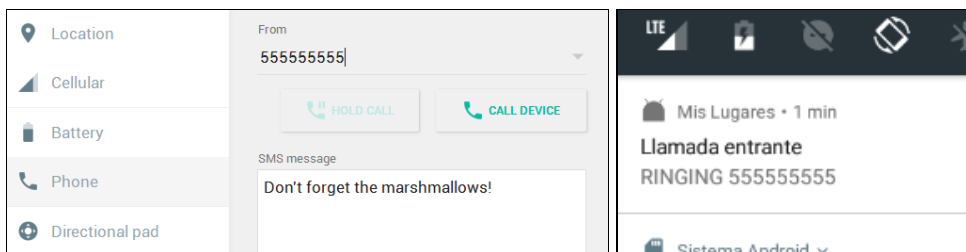
² <https://developer.android.com/guide/components/broadcast-exceptions?hl=es-419>


```

        NotificationCompat.Builder notificacion = new
            NotificationCompat.Builder(context)
                .setContentTitle("Llamada entrante ")
                .setContentText(info)
                .setSmallIcon(R.mipmap.ic_launcher)
                .setContentIntent(PendingIntent.getActivity(context, 0,
                    new Intent(context, MainActivity.class), 0));
        ((NotificationManager) context.getSystemService(Context.
            NOTIFICATION_SERVICE)).notify(1,notificacion.build());
    }
}
}
}

```

5. Ejecuta la aplicación e introduce una llamada. Si trabajas con un dispositivo de versión 6 o superior debes dar permiso de teléfono a la aplicación manualmente. Si utilizas el emulador, puedes utilizar los tres puntos que aparecen en la parte inferior de la barra de herramientas, para abrir los controles extendidos. Selecciona *Phone*, introduce un número de teléfono y pulsa *CALL DEVICE*.



6. Verifica que se crea la notificación.
7. Abre la lista de notificaciones y verifica la información mostrada:



Recursos adicionales: *Lista de anuncios broadcast*

La siguiente lista muestra los anuncios *broadcast* más importantes organizados por temas. La columna de la izquierda muestra la constante en Java que identifica la acción *broadcast*. En la columna de la derecha, (*No Manifest*): indica que no se puede declarar el receptor de anuncios en *AndroidManifest.xml*. Solo se puede utilizar `registerReceiver()`. (*Solo sistema*): indica que se trata de una intención protegida, que solo puede ser lanzada por el sistema. También se indica si se requiere de algún permiso especial y si hay información *EXTRA* que se pasa a través de la intención.

Nombre de la acción / (<u>CONSTANTE</u>)	Descripción / Permiso (<u>INFORMACIÓN</u> <u>EXTRA EN INTENT</u>)
---	--

Batería

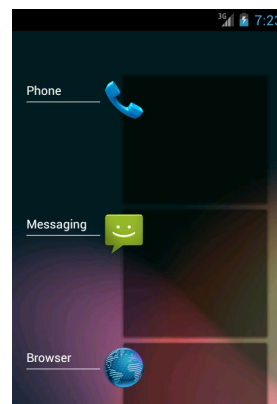
Nombre de la acción / (CONSTANTE)	Descripción / Permiso (INFORMACIÓN EXTRA EN INTENT)
android.intent.action.BATTERY_LOW (ACTION_BATTERY_LOW)	Batería baja (<i>Solo sistema</i>).
android.intent.action.BATTERY_OKAY (ACTION_BATTERY_OKAY)	Batería correcta después de haber estado baja (<i>Solo sistema</i>).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

Nombre de la acción / (CONSTANTE)	Descripción / Permiso (INFORMACIÓN EXTRA EN INTENT)
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 (<i>Solo sistema</i>).
Memoria y escáner multimedia	
android.intent.action.DEVICE_STORAGE_LOW (ACTION_DEVICE_STORAGE_LOW)	Queda poca memoria (<i>Solo sistema</i>).
android.intent.action.DEVICE_STORAGE_OK (ACTION_DEVICE_STORAGE_OK)	Salimos de la condición de poca memoria (<i>Solo sistema</i>).
android.intent.action.MEDIA_EJECT (ACTION_MEDIA_EJECT)	El usuario pide extraer almacenamiento externo.
android.intent.action.MEDIA_MOUNTED (ACTION_MEDIA_MOUNTED)	Almacenamiento externo disponible.
android.intent.action.MEDIA_REMOVED (ACTION_MEDIA_REMOVED)	Almacenamiento externo 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 versión de tu aplicación ha sido instalada (<i>Solo sistema</i>).
android.intent.action.PACKAGE_ADDED (ACTION_PACKAGE_ADDED)	Una nueva aplicación instalada (EXTRA_UID , EXTRA_REPLACING) (<i>Solo sistema</i>).
android.intent.action.PACKAGE_FIRST_LAUNCH (ACTION_PACKAGE_FIRST_LAUNCH)	Primera vez que se lanza una aplicación (<i>Solo sistema</i>).
android.intent.action.PACKAGE_REMOVED (ACTION_PACKAGE_REMOVED)	Se desinstala una aplicación (<i>Solo sistema</i>).

Nombre de la acción / (<u>CONSTANTE</u>)	Descripción / Permiso (<u>INFORMACIÓN</u> <u>EXTRA EN INTENT</u>)
Comunicaciones y redes	
<code>android.intent.action.PHONE_STATE</code> (<u><code>ACTION_PHONE_STATE_CHANGED</code></u>)	Cambia el estado del teléfono. P. ej., hay una llamada. Permiso: <u><code>READ_PHONE_STATE</code></u> (<u><code>EXTRA_STATE</code></u> , <u><code>EXTRA_STATE_RINGING</code></u>).
<code>android.intent.action.NEW_OUTGOING_CALL</code> (<u><code>ACTION_NEW_OUTGOING_CALL</code></u>)	Se va a hacer una llamada. Permiso <u><code>PROCESS_OUTGOING_CALLS</code></u> (<u><code>EXTRA_PHONE_NUMBER</code></u>) (<i>Solo sistema</i>).
<code>android.provider.Telephony.SMS_RECEIVED</code>	Se recibe un SMS. Permiso: <u><code>RECEIVE_SMS</code></u> (Extra: " <i>pdus</i> " ver ejemplo ³).
<code>android.bluetooth.adapter.action.DISCOVERY_STARTED</code> (<u><code>ACTION_DISCOVERY_STARTED</code></u>)	Comienza escáner Bluetooth.
<code>android.bluetooth.adapter.action.STATE_CHANGED</code> (<u><code>ACTION_STATE_CHANGED</code></u>)	Bluetooth habilitado/deshabilitado.
<code>android.net.wifi.NETWORK_IDS_CHANGED</code> (<u><code>NETWORK_IDS_CHANGED_ACTION</code></u>)	Cambia el identificador de la red Wi-Fi conectada.
<code>android.net.wifi.STATE_CHANGE</code> (<u><code>NETWORK_STATE_CHANGED_ACTION</code></u>)	Cambia la conectividad Wi-Fi (<u><code>EXTRA_NETWORK_INFO</code></u> , <u><code>EXTRA_BSSID</code></u> , <u><code>EXTRA_WIFI_INFO</code></u>).
<code>android.net.wifi.RSSI_CHANGED</code> (<u><code>RSSI_CHANGED_ACTION</code></u>)	Cambia el nivel de señal Wi-Fi (<u><code>EXTRA_NEW_RSSI</code></u>).

6.5.2. Arrancar una actividad en una nueva tarea desde un receptor de anuncio

El concepto de tarea no había sido introducido en el curso. Sin embargo, resulta sencillo, y seguro que si eres usuario de Android estás familiarizado con él. La forma más sencilla de entenderlo es que pulses en tu dispositivo móvil el botón cuadrado si tienes la versión 5.0 o superior o el Casa durante un segundo, en versiones anteriores. Se mostrará la lista de tareas que hay actualmente en ejecución o que han sido ejecutadas recientemente.



³ stackoverflow.com/questions/33517461/smsmessage-createfrompdu-is-deprecated-in-android-api-level-23

Puedes intercambiar de tarea simplemente pulsando sobre una de las previsualizaciones que aparecen en pantalla.

No hay que confundir el concepto de tarea con el de aplicación. Para iniciar una nueva tarea puedes pulsar al botón de Casa y pulsar sobre uno de los iconos de la pantalla inicial. De esta forma se iniciará la aplicación correspondiente (por ejemplo, el lector de correo). Desde una tarea se pueden arrancar nuevas aplicaciones; por ejemplo, desde un correo podemos acceder a una URL ejecutando el navegador web. Esta nueva aplicación se ejecutará en la misma tarea.

Otro aspecto a destacar es que cada tarea tiene una pila de actividades independiente. Es decir, si pulsamos el botón de volver en la tarea descrita, pasaremos de nuevo al lector de correo. Pero si cambiamos de tarea y pulsamos el botón de volver, el resultado será muy diferente.



Ejercicio: Arranque de una actividad al llegar un SMS

Vamos a modificar el proyecto Asteroides para que se arranque automáticamente la actividad `AcercaDeActivity` al llegar un SMS cualquiera.

1. Abre el proyecto Asteroides.
2. En *AndroidManifest.xml* pide el permiso adecuado y registra el receptor de anuncios:

```
...
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
...
<application>
    ...
    <receiver android:name="ReceptorSMS" >
        <intent-filter>
            <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
        </intent-filter>
    </receiver>
</application>
...
```

3. Crea una nueva clase con el siguiente código:

```
public class ReceptorSMS extends BroadcastReceiver {
    @Override public void onReceive(Context context, Intent intent) {
        Intent i = new Intent(context, AcercaDeActivity.class);
        i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        context.startActivity(i);
    }
}
```

La forma de arrancar una actividad desde un receptor de anuncios es muy similar a la que hemos estudiado en el capítulo 3. La única diferencia es que

ahora hemos necesitado añadir un *flag* a la intención, que indica que la actividad sea creada en una nueva tarea.

4. Ejecuta la aplicación. Envía un SMS al dispositivo y verifica que se abre la actividad *Acerca de* ...

NOTA: Si utilizas el emulador, puedes utilizar la vista EmulatorControl de Android Studio para simular el envío de un SMS.

Cuando lanzamos una nueva actividad, Android nos permite controlar en qué tarea y en qué posición de la pila se situará. No obstante, se recomienda usar siempre el sistema estándar. Es decir, si lanzamos una nueva actividad desde otra actividad, la nueva actividad se sitúa en la misma tarea en la cima de la pila de actividades. Otra cosa es lanzar la actividad desde un receptor de anuncios, dado que cuando llegue el SMS podemos encontrarnos en cualquier tarea. En ese caso, resulta imprescindible activar el *flag* `FLAG_ACTIVITY_NEW_TASK`, así podrá crearse una nueva tarea.



Enlaces de interés:

Lanzar las actividades de la forma estándar suele ser lo más adecuado en la mayoría de los casos. No obstante, si quieres profundizar sobre este tema te recomendamos los siguientes enlaces:

- **Tasks and Back Stack:** Documentación oficial de Android.
<http://developer.android.com/guide/components/tasks-and-back-stack.html>
- **Manipulating Android tasks and back stack:** Presentación didáctica con muchos ejemplos.
<http://es.slideshare.net/RanNachmany/manipulating-android-tasks-and-back-stack>

6.5.3. Arrancar un servicio tras cargar el sistema operativo

En muchas ocasiones puede ser interesante que un servicio de nuestra aplicación esté siempre activo, incluso aunque el usuario no haya arrancado nuestra aplicación. Imagina, por ejemplo, un servicio de mensajería instantánea, que ha de estar siempre atento a la llegada de mensajes. Conseguirlo es muy fácil, no tenemos más que crear un receptor de anuncios que se active ante el anuncio `android.intent.action.BOOT_COMPLETED`. Desde este receptor podremos crear el servicio. Para poder registrar el receptor es obligatorio solicitar el permiso `RECEIVE_BOOT_COMPLETED`. Veamos los tres pasos a seguir:

1. Creamos un receptor de anuncios:

```
public class ReceptorArranque extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        context.startService(new Intent(context, Servicio.class));
    }
}
```

2. Creamos el servicio:

```
public class Servicio extends Service {
    @Override
    public void onCreate() {...}
    @Override
    public int onStartCommand(Intent intencion, int flags,
        int idArran){...}
    @Override
    public void onDestroy() {...}
    @Override
    public IBinder onBind(Intent intencion) {
        return null;
    }
}
```

3. En *AndroidManifest.xml* pedimos el permiso adecuado y registramos el receptor de anuncios:

```
...
<uses-permission android:name =
    "android.permission.RECEIVE_BOOT_COMPLETED"/>
<application>
...
    <receiver android:name="ReceptorArranque" >
        <intent-filter >
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
    </receiver>
</application>
...
```

NOTA: Si estás trabajando con un emulador, puedes lanzar el anuncio *BOOT_COMPLETED* de un modo rápido y sin necesidad de reiniciar el emulador. Para ello, ejecuta el siguiente comando en el terminal del sistema (o desde la ventana terminal de Android Studio). Solo funciona **en emuladores con API<24**.

```
adb shell am broadcast -a android.intent.action.BOOT_COMPLETED
```

NOTA: Solo las aplicaciones instaladas en memoria interna reciben *BOOT_COMPLETED*. Para forzar que tu aplicación se instale en memoria interna añade el siguiente atributo en la etiqueta *<manifest>*:

```
Android:installLocation="internalOnly"
```



Práctica: Arranque automático del servicio de música

Modifica el proyecto *ServicioMusica* para que el servicio se active desde el arranque del sistema operativo.

6.5.4. Anuncios *broadcast* permanentes

Android permite enviar dos tipos de anuncios *broadcast*, normales y permanentes. Un *broadcast* permanente llegará a los receptores de anuncios que actualmente estén escuchando, pero también a los que se instancien en un futuro. Por ejemplo, el sistema emite el anuncio *broadcast* `ACTION_BATTERY_CHANGED` de forma permanente. De esta forma, cuando se llama a `registerReceiver()` se obtiene la intención de la última emisión de este anuncio. Por lo tanto, puede usarse para encontrar el estado de la batería sin necesidad de esperar a un futuro cambio en su estado. Este tipo de anuncios también se conocen como persistentes o pegajosos (del término en inglés *sticky*).

Para enviar un *broadcast* permanente utiliza el método `sendStickyBroadcast()` en lugar de `sendBroadcast()`:

```
Intent i = new Intent("com.example.intent.action.ACTION");  
sendStickyBroadcast(i);
```

Enviar un anuncio permanente requiere de la solicitud del siguiente permiso:

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

Se exige este permiso dado que las aplicaciones mal intencionadas pueden ralentizar el dispositivo o volverlo inestable al demandar demasiada memoria.



Preguntas de repaso: *Receptores de anuncios*

6.6. Un receptor de anuncios como mecanismo de comunicación

Hasta ahora hemos visto como los receptores de anuncios nos permitían reaccionar ante ciertas circunstancias que ocurrían en el sistema (batería baja, llamada entrante, etc.). En este apartado vamos a ver lo sencillo que resulta crear nuestros propios anuncios *broadcast* y recogerlos desde cualquier componente de nuestra aplicación. Además, estos anuncios también podrán recogerse desde otras aplicaciones.

En un apartado anterior hemos visto cómo asociar anuncios *broadcast* a receptores de anuncios por medio de *AndroidManifest.xml*. En este apartado vamos a realizar la misma tarea utilizando Java. El programador puede escoger uno u otro modo según le convenga.



Ejercicio: *Creación de un nuevo tipo de anuncio broadcast*

En el ejercicio anterior hemos creado un servicio desde una actividad para realizar una operación matemática. Una vez que el servicio ha concluido la

operación, queremos que avise a la actividad y le devuelva el valor calculado. En este ejercicio realizaremos este trabajo por medio de un anuncio *broadcast*.

1. Abre el proyecto `IntentService` creado en el apartado anterior.
2. Añade el siguiente código dentro de la clase `MainActivity`:

```
public class ReceptorOperacion extends BroadcastReceiver {  
    public static final String ACTION_RESP =  
        "com.example.intentservice.intent.action.ESPUESTA_OPERACION";  
  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        Double res = intent.getDoubleExtra("resultado", 0.0);  
        salida.append(" " + res + "\n");  
    }  
}
```

Esta nueva clase solo va a utilizarse en esta actividad, por lo que puede definirse dentro de la clase `MainActivity`, en lugar de en un fichero independiente. Se trata de un receptor *broadcast*, que cada vez que llegue un nuevo anuncio leerá un valor enviado en el extra `"resultado"` y lo añadirá al `TextView salida`.

3. Añade las siguientes líneas al método `onCreate()`:

```
IntentFilter filtro = new IntentFilter(ReceptorOperacion.ACTION_RESP);  
filtro.addCategory(Intent.CATEGORY_DEFAULT);  
registerReceiver(new ReceptorOperacion(), filtro);
```

Con este código hemos asociado el tipo de anuncio *broadcast* a nuestro receptor de anuncios. Como hemos visto en otro apartado, esta tarea también puede realizarse por medio de *AndroidManifest.xml*. El programador puede escoger uno u otro modo según le convenga. Al tratarse de un anuncio para una comunicación interna a nuestra aplicación, parece más conveniente realizarlo así que publicarlo por *AndroidManifest.xml*.

4. Nos queda lanzar el anuncio *broadcast*. Para ello reemplaza la siguiente línea de `IntentServiceOperacion.onHandleIntent()`:

```
MainActivity.salida.append(n*n + "\n");
```

por:

```
Intent i = new Intent();  
i.setAction(MainActivity.ReceptorOperacion.ACTION_RESP);  
i.addCategory(Intent.CATEGORY_DEFAULT);  
i.putExtra("resultado", n*n);  
sendBroadcast(i);
```

5. Verifica que la aplicación funciona perfectamente. Pulsa repetidas veces el botón y verifica que esta no se bloquea mientras se calculan las operaciones. Advierte como, aunque se pulse tres veces seguidas, no comienzan las tres operaciones a la vez. Estas serán realizadas de una en una, de manera que irán apareciendo los resultados a intervalos de 5 segundos.

6. Modifica el tiempo de retardo para que este sea de 25 seg. (`sleep(25000)`). Ejecuta de nuevo la aplicación y observa como el sistema no nos muestra ningún error.



Preguntas de repaso: *Receptores anuncios para la comunicación*

6.7. Un servicio como mecanismo de comunicación entre aplicaciones

Como hemos comentado, un servicio tiene una doble funcionalidad: además de permitir la ejecución de código en segundo plano, vamos a poder utilizarlo como un mecanismo de comunicación entre aplicaciones⁴. Cuando una aplicación quiere compartir algún tipo de información con otra aplicación, se presenta un problema. Las aplicaciones en Android se ejecutan en procesos separados y, por tanto, tienen espacios de memoria distintos. Esto nos impide, por ejemplo, que ambas aplicaciones compartan un mismo objeto.

Como respuesta a este problema, Android nos propone un mecanismo de comunicación entre procesos que se basa en un lenguaje de especificación de interfaces, AIDL (*Android Interface Definition Language*). Las interfaces AIDL se publican por medio de servicios. Gracias al lenguaje de especificación de interfaces AIDL, un proceso en Android puede llamar a un método de un objeto situado en un proceso diferente del suyo. Se trata de un mecanismo de comunicación entre procesos similar a COM o Corba, aunque algo más ligero.

Si queremos comunicar dos aplicaciones a través de este mecanismo, seguiremos los siguientes pasos:

1. Escribiremos un fichero AIDL: En él se define la interfaz, es decir, los métodos y los parámetros que luego podremos utilizar.
2. Implementaremos los métodos de la interfaz: Para ello habrá que crear una clase en Java que implemente estos métodos.
3. Publicar la interfaz a los clientes: Para ello se extenderá la clase `Service` y sobrescribiremos el método `onBind(Intent)` de forma que devuelva una instancia de la clase que implementa la interfaz.



Vídeo[tutorial]: *Un servicio como mecanismo de comunicación entre aplicaciones*

⁴ <http://developer.android.com/guide/topics/fundamentals.html#rpc>

Veamos estos tres pasos más detenidamente por medio de un ejemplo. Para ello crea la siguiente aplicación:

```
Application Name: Servicio Remoto
Package Name: org.example.servicioremoto
☒ Phone and Tablet
Minimum SDK: API 15 Android 4.0.3 (IceCreamSandwich)
```

Reemplaza el código del *layout* `activity_main.xml` por el mismo utilizado en `ServicioMusica`. Reemplaza los textos de los botones *Arrancar servicio* por *Conectar servicio* y *Detener servicio* por *Desconectar servicio*. Crea dos botones más. Uno con texto “Reproducir” e *id* “@+id/boton_reproducir” y otro con texto “Avanzar” e *id* “@+id/boton_avanzar”.

Copia el fichero `res/raw/audio.mp3` en la nueva aplicación.