Servicios (1ª parte)

Contenidos

Componentes de una aplicación

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

Servicio

Los servicios son componentes que se ejecutan en background (o segundo plano) y que no interactúan con el usuario.
Se utilizan para operaciones repetitivas y potencialmente largas en ejecución, es decir, descargas de Internet, comprobación de nuevos datos, procesamiento de datos, actualización de los proveedores de contenidos, reproducción de música y similares.
Funcionan con una prioridad más alta que las actividades inactivas o invisibles. Además también se pueden configurar para reiniciarse si son terminados por el sistema.
Desde Ajustes->Aplicaciones (Android<6) o Ajustes->Desarrollador (Android6+) podemos ver los servicios en ejecución en el dispositivo y forzar su cierre.
Segundo plano no implica por defecto que sea un proceso separado ni que sea un hilo secundario.
Esto significa que si el servicio va a hacer algún trabajo intensivo de la CPU o puede quedar bloqueado en ciertas operaciones (como la reproducción de MP3 en red), debe crearse un nuevo hilo/tarea asíncrona para hacer el trabajo del servicio y se reducirá el riesgo de que la aplicación no responda (ANR) y el hilo principal de la aplicación puede seguir estando dedicado a la interacción con el usuario en sus actividades. Además puede utilizarse la clase IntentService para lanzar un servicio en su propio hilo.
También puede especificarse que se ejecuta en un proceso independiente a través del valor del atributo android:process=":process_description".

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 la aplicación.
Para 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 declarar el permiso RECEIVE_BOOT_COMPLETED.

Tipos de servicios

Servicios del sistema

- Android ofrece una gran cantidad de servicios predefinidos, disponibles regularmente a través de la clase Manager y el método getSystemService()
- Ya conocemos muchos de ellos, por ejemplo:
  NotificationManager notificationManager = (NotificationManager) getSystemService (***NOTIFICATION_SERVICE***);

public abstract Object getSystemService (String name)

Return the handle to a system-level service by name. The class of the returned object varies by the requested name. Currently available names are:

WINDOW_SERVICE ("window")
The top-level window manager in which you can place custom windows. The returned object is a WindowManager.

LAYOUT_INFLATER_SERVICE ("layout_inflater")
A LayoutInflater for inflating layout resources in this context.

ACTIVITY_SERVICE ("activity")
A ActivityManager for interacting with the global activity state of the system.

POWER_SERVICE ("power")
A PowerManager for controlling power management.

ALARM_SERVICE ("alarm")
A AlarmManager for receiving intents at the time of your choosing.

NOTIFICATION_SERVICE ("notification")
A NotificationManager for informing the user of background events.

KEYGUARD_SERVICE ("keyguard")
A KeyguardManager for controlling keyguard.

LOCATION_SERVICE ("location")
A LocationManager for controlling location (e.g., GPS) updates.

SEARCH_SERVICE ("search")
A SearchManager for handling search.

VIBRATOR_SERVICE ("vibrator")
A Vibrator for interacting with the vibrator hardware.

CONNECTIVITY_SERVICE ("connection")
A ConnectivityManager for handling management of network connections.

WIFI_SERVICE ("wifi")
A WifiManager for management of Wi-Fi connectivity.

WIFI_P2P_SERVICE ("wifip2p")
A WifiP2pManager for management of Wi-Fi Direct connectivity.

INPUT_METHOD_SERVICE ("input_method")
An InputMethodManager for management of input methods.

UI_MODE_SERVICE ("uimode")
An UiModeManager for controlling UI modes.

DOWNLOAD_SERVICE ("download")
A DownloadManager for requesting HTTP downloads

BATTERY_SERVICE ("batterymanager")
A BatteryManager for managing battery state

JOB_SCHEDULER_SERVICE ("taskmanager")
A JobScheduler for managing scheduled tasks

Servicios propios

- Deben ser declarados en el archivo AndroidManifest.xml. El único atributo obligatorio es "android:name".

```
<application .... >
    ....
    ....

    <service android:name="com.example.services.MiService1"/>

    <service android:name="com.example.services.MiService2">
        ....
        ....
    </service>

</application>
```

- Pueden definirse otras propiedades: más información (la ya comentada *android:process* o *android:exported* para fijar el servicio como propio solo de la aplicación, etc.).

```
<service
        android:name="MyService"
        android:icon="@drawable/icon"
        android:label="@string/service_name"
        >
</service>
```

Servicios del sistema

AlarmManager

En determinados tipos de aplicaciones es necesario ejecutar en instantes de tiempo futuros determinadas acciones, incluso aunque la aplicación no esté ejecutándose en ese momento, aquí entra en juego el servicio de alarmas de Android, gestionado por la clase AlarmManager.

Gracias a este servicio, el programador podrá registrar con alarmManager.**set**() (o con alarmManager.exact() si se se desea precisión como en los despertadores) y cancelar con alarmManager.**cancel**() el deseo de ejecutar acciones (a través del uso de Intent) en un determinado instante.

El Intent (o PendingIntent) puede ser para lanzar un BroadCast (PendingIntent.getBroadcast) o para lanzar una actividad (PendingIntent.getActivity). Además el BroadCast puede iniciar un Servicio si es necesario.

Proyecto P_79_ServicioAlarma

Se desea que pasados 5 segundos de lanzar la aplicación el móvil vibre durante 10 segundos (para qué?: para aprender!).

La acción de vibrar estará gestionada en un Broadcast llamado MyReceiver que pone en marcha el servicio VIBRATOR_SERVICE.

A dicho Broadcast se accederá por Intent y PendingIntent desde la MainActivity que también pone en marcha el servicio de Alarmas y lanza el cronómetro para los 5 segundos.

En el Androidmanifest es obligatorio fijar el permiso <uses-permission android:name="android.permission.VIBRATE" /> además de, obviamente, declarar actividad y receiver. El servicio no se declara por ser del sistema.

(Nota: para simplificar código, la MainActivity del proyecto de esta unidad utiliza como plantilla de AS la de EmptyActivity).

Clase MainActivity

```
public class MainActivity extends AppCompatActivity {

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

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action",
Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });

        Intent intent = new Intent(this, MyReceiver.class);
        PendingIntent pendingIntent =
PendingIntent.getBroadcast(this.getApplicationContext(), 123, intent,
PendingIntent.FLAG_CANCEL_CURRENT);
        AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
        if (alarmManager != null) {
            alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis()+ (5 *
1000), pendingIntent);
        }

    }
```

Clase MyReceiver

```
public class MyReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        //Define la vibracion del telefono durante 10 segundos
        Vibrator vibrator = (Vibrator)
context.getSystemService(Context.VIBRATOR_SERVICE);
        vibrator.vibrate(10000);
        Toast.makeText(context,"Vibro",Toast.LENGTH_SHORT).show();
    }
}
```

JobScheduler

AlarmManager puede usarse para programar tareas pero con la desventaja de que el gestor de alarmas no es consciente de la situación actual del dispositivo, por ejemplo, no "sabe" si el dispositivo está conectado a un enchufe, en reposo o conectado a un determinado tipo de red que a lo mejor es cuando interesa que se realicen las tareas.
Desde la API 21 (Android 5) se puede usar la clase JobScheduler ya que este planificador de tareas permite tener en cuenta el estado del dispositivo
En este curso no trabajaremos con programadores de tareas de este tipo, pero hay excelentes ejemplos de su uso en muchos enlaces, como por ejemplo Scheduling of tasks with the Android JobScheduler – Tutorial o Android Development Tutorial: Job Scheduler.
Android 7 también ofrece optimizaciones.

Servicios propios

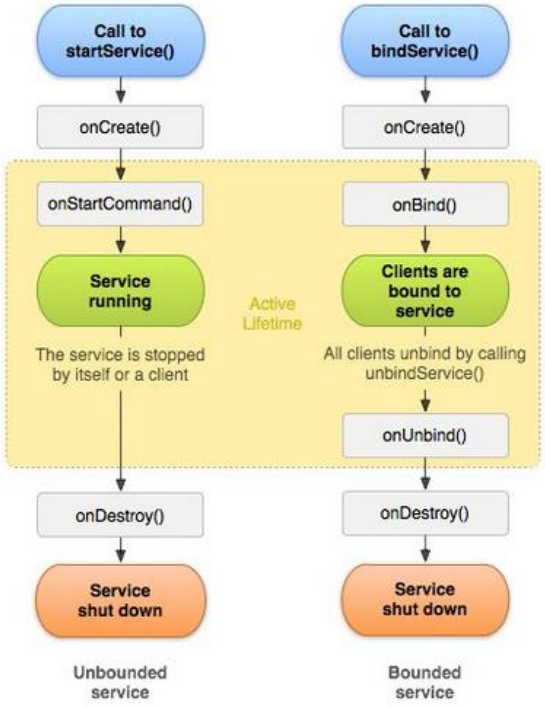
Android distingue dos tipos de servicios:

- **Servicios locales (o iniciados)**: Servicios que forman parte de la aplicación y solo accesibles desde ella (aunque pueden ser lanzados por otras aplicaciones). Las actividades/receptores de anuncios que los usen harán uso de las llamadas **Context.startService()** para lanzarlos y **Context.stopService()** para detenerlos.
- **Servicio vinculados (o remotos o enlazados)**: Servicios que están accesibles también por otras aplicaciones instaladas en el dispositivo. Para usarlos se hace uso de **Context.bindService()** y **Context.unbindService()**.

Sean del tipo que sean deben declararse en el AndroidManifest.
Se inician a partir de otros componentes Android: actividades, content provider, broadcast y/o otros servicios

Ciclo de vida de un servicio y métodos según tipo

Un método común para los dos tipos de servicios que es ***onCreate()***. Normalmente configuraremos aquí nuestro servicio.



Los servicios "Started" (locales) se inician de forma indefinida en el método ***onStartCommand()***. Para detener el servicio deberemos llamar a ***stopSelf()*** o ***stopSelfResult()*** desde el mismo servicio o a ***stopService()*** desde la activity o componente que lo puso en marcha.

En los servicios "Bound" (vinculados) se inicia su vida en el método ***onBind()*** que será el encargado de enlazar el servicio con la activity o componente que lo llama. Se ejecutara de forma indefinida hasta que llamemos a ***unbindService()*** desde la activity o componente que lo puso en marcha, de esta manera se cierra la conexión y se terminar el servicio, en este momento el servicio pasa al método ***onUnbind()***.

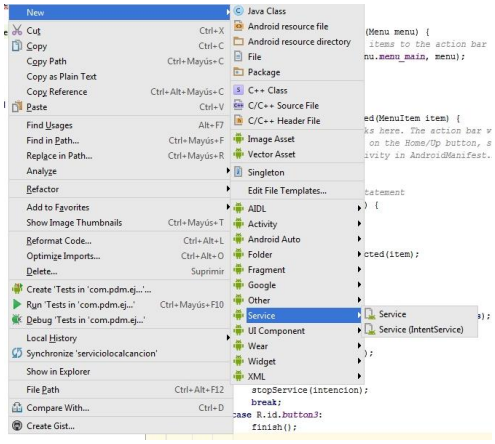
Otro método común para los dos tipos de servicios es ***onDestroy()***. Sera el lugar donde guardaremos los datos importantes y limpiaremos el servicio de los recursos que hayamos podido utilizar.

Servicio local

Podríamos dividirlos en dos grupos, los que extienden de la clase "Service" y los que extienden de la clase "IntentService":

- Service: es la clase base para crear cualquier servicio. Una vez iniciado el servicio se ejecutara de forma indefinida en segundo plano. Si la tarea es muy pesada, es importante crear un hilo para hacer todo el trabajo.
- IntentService: es una subclase de Service que maneja las peticiones una a una. Ideal si no necesitamos manejar múltiples peticiones al mismo tiempo. Una vez acabe de ejecutar la última petición automáticamente el servicio termina.

AS nos proporciona asistente para la creación de ambos que incluye plantilla de código y declaración en Manifest.



Servicio local de tipo Service: P_80_ServicioLocal_1

El proyecto P_80_ServicioLocal_1 tiene una actividad con 3 botones, uno para arrancar música de fondo (y así empezamos a usar la clase MediaPlayer) que reproducirá la música guardada en la carpeta res/raw en el fichero audio.mp3, otro que parará la reproducción y el último que detendrá la app (y así comprobaremos que la música sigue reproduciéndose).

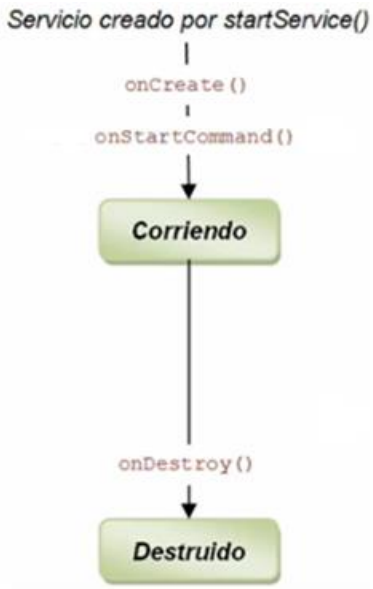
Código MainActivity

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button iniciar = findViewById(R.id.button);
        Button parar = findViewById(R.id.button2);
        Button acabar = findViewById(R.id.button3);
        final Intent intent = new Intent(this, MusicaFondo.class);
        iniciar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                startService(intent);
            }
        });
        parar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                stopService(intent);
            }
        });
        acabar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                finish();
            }
        });
    }
}
```

Obviamente, al intent le podríamos haber añadido datos (con putExtra) si los necesita el servicio.

Código MusicaFondo:

```
public class MusicaFondo extends Service {
    MediaPlayer reproductor;

    public MusicaFondo() {
    }

    @Override
    public void onCreate() {
        reproductor = MediaPlayer.create(this, R.raw.audio);
        reproductor.setLooping(true);
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int
idArranque) {
        reproductor.start();
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        reproductor.stop();
    }

    @Override
    public IBinder onBind(Intent intent) {
        // Este método permite tener acceso a los miembros y métodos en el interior del
Service.
        // Como es de tipo local no lo usamos
        return null;
    }
}
```



El método más importante es onStartCommand():

Parámetros de entrada (flags e idArranque son generados por el sistema)	Devuelve cómo debe comportarse el sistema cuando el proceso donde está el servicio se ha destruido (normalmente en situaciones de baja memoria)
`public int onStartCommand (Intent intencion, int flags, int idArranque)` `intencion` Un objeto `Intent` que se indicó en la llamada `startService(Intent)`. `flags` Información sobre como comienza la solicitud. Puede ser `0`, `START_FLAG_REDELIVERY` o `START_FLAG_RETRY`. Un valor distinto de 0 se utiliza cuando se reinicia un servicio tras haber sido interrumpido. `idArranque` Un entero único representando la solicitud de arranque específica. Usar este mismo estero en el método `stopSelfResult(int idArranque)`.	`START_STICKY`: El sistema tratará de recrear el servicio, se realizará una llamada a `onStartCommand()` pero con el parámetro `intent` igual a `null`. Usar cuando el servicio puede arrancar sin información adicional. `START_NOT_STICKY`: El sistema no tratará de volver a crear el servicio, por lo tanto el parámetro `intent` nunca podrá ser igual a `null`. Usar cuando el servicio no puede reanudarse una vez interrumpido. `START_REDELIVER_INTENT`: El sistema tratará de volver a crear el servicio. El parámetro `intent` será el que se utilizó en la última llamada `startService(Intent)`.

- Si se producen varias llamadas a startService() no supondrá la creación de varios servicios, aunque sí que se realizarán múltiples llamadas a onStartCommand(). No importa cuántas veces el servicio haya sido creado, parará con la primera invocación de stopService() o stopSelf(). Sin embargo, podemos utilizar el método stopSelf(int startId) para asegurarnos que el servicio no parará hasta que todas las llamadas hayan sido procesadas.
- En situaciones donde el sistema necesite memoria conservar un servicio siempre será menos prioritario que la actividad visible en pantalla, 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. Prioridad para limpiar procesos en situaciones de baja memoria
- Si la tarea fuese muy larga, es en este método donde se llamaría al hilo/tarea asíncrona que la realizase.

Método **onBind()**

Si la actividad quiere interactuar con el servicio podemos utilizar el método **bindService()**, que requiere de un objeto **ServiceConnection** que nos permitirá conectarnos con el servicio y que retorna un objeto **IBinder**, el cuál puede ser utilizado para que la actividad pueda comunicarse con el servicio. Este método siempre debe implementarse, incluso dentro de los "started services", en cuyo caso devuelve null.
(Se explica en servicios vinculados ya que es tratar al servicio local como vinculado, siendo el cliente la propia actividad que llama al servicio).

Servicio local de tipo IntentService: P_81_ServicioLocal_2

Es un tipo particular de servicio Android que se preocupará por nosotros de la creación y gestión del nuevo hilo de ejecución y de detenerse a sí mismo una vez concluida su tarea asociada.

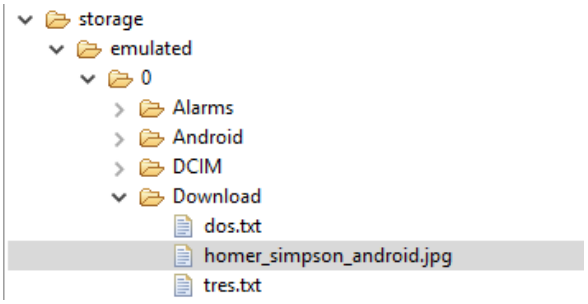
La utilización de un IntentService va a ser tan sencilla como extender una nueva clase de IntentService e implementar su método **onHandleIntent()** que será lanzado cada vez que se arranque el servicio en un hilo nuevo. Este método recibe como parámetro un Intent, que podremos utilizar para pasar al servicio los datos de entrada necesarios y será el que contenga el código de la tarea a ejecutar en segundo plano.

Limitaciones:

- No puede interactuar directamente con la interfaz de usuario (hay que usar por ejemplo un BroadCast Receiver para lanzar Toast, progressBar,…).
- Las peticiones de trabajo se ejecutan secuencialmente. Si una operación se está ejecutando en un IntentService, y se envía otra petición, la solicitud espera hasta que se termina la primera operación. Se irán atendiendo una tras otra sin que haya dos a la vez en ejecución.
- Una operación que se ejecuta en un IntentService no puede ser interrumpida.

El proyecto P_81_ServicioLocal_2 descarga una imagen de la web utilizando este tipo de servicio de la siguiente manera:

- Actividad MainActivity: Botón para iniciar servicio (debería mejorarse porque siempre se descarga lo mismo ☺!)
- Servicio DescargaServicio: en el método onHandleIntent() recibe url y nombre del fichero y se encarga de la descarga. Llama al Broadcast enviando información sobre si la descarga ha sido correcta o fallida.
- Broadcast ReceiverFinDescarga: informa del resultado de la descarga mediante Toast.
- Permisos Manifest:
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
- Nota: la aplicación hace uso de permiso peligroso, por tanto, hay que tratarlo en código. Como otras veces, crearemos nuestros metodos seguir() y pedirpermiso() y sobreescribiremos onRequestPermissionsResult. Pero estos dos últimos hacen uso de la vista Snackbar, si has partido de la plantilla EmptyActivity no está incluida la biblioteca necesaria, recuerda añadir en el fichero build.gradle de la app su dependencia.
- Advertencia: el código está escrito para guardar la imagen en la carpeta DOWNLOAD (que algunos emuladores no tienen!)



Código MainActivity:

```
public class MainActivity extends AppCompatActivity {

    private View layoutMain;
    private ReceiverFinDescarga receiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        layoutMain = findViewById(R.id.activity_main);
        Button button = findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if (ContextCompat.checkSelfPermission(getApplicationContext(),
Manifest.permission.WRITE_EXTERNAL_STORAGE) == PackageManager.PERMISSION_GRANTED) {
                    seguir();
                } else {
                    pedirPermiso();
                }
            }
        });
    }
    private void pedirPermiso() {
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
Manifest.permission.WRITE_EXTERNAL_STORAGE)) {
            final Activity activity = this;
            Snackbar.make(layoutMain, "Sin el permiso escritura en SD no puedo
escribir.", Snackbar.LENGTH_INDEFINITE)
                    .setAction("OK", new View.OnClickListener() {
                        @Override
                        public void onClick(View view) {
                            ActivityCompat.requestPermissions(activity, new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}, 123);
                        }
                    })
                    .show();
        } else {
            ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}, 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) {
                seguir();
            } else {
                Snackbar.make(layoutMain, "Sin el permiso, no puedo realizar la acción",
Snackbar.LENGTH_LONG).show();
            }
        }
    }

    private void seguir() {
        // Registramos el receiver que se usará posteriormente en el servicio
        // Como solo lo utiliza esta app usamos la clase LocalBroadcastManager
        receiver = new ReceiverFinDescarga();
        LocalBroadcastManager.getInstance(this).registerReceiver(receiver, new
IntentFilter("descargaCompletada"));
        // Lanzamos el servicio con los datos necesarios añadidos al intent
        Intent intent = new Intent(getApplicationContext(), DescargaServicio.class);
        intent.putExtra("urlDescarga",
"https://eandroid.files.wordpress.com/2012/07/homer_simpson_android.jpg");
        startService(intent);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Desregistramos el receiver
        LocalBroadcastManager.getInstance(this).unregisterReceiver(receiver);
    }
}
```

Código DescargaServicio

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

    @Override
    protected void onHandleIntent(Intent intent) {
        if (intent != null) {
            int result = Activity.RESULT_CANCELED;
            String stringUrl = intent.getStringExtra("urlDescarga");
            String nombreFichero = stringUrl.substring(stringUrl.lastIndexOf('/') + 1);
            File ficheroDescargado = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS),
nombreFichero);
            FileOutputStream escritura = null;
            InputStream lectura = null;
            try {
                URL url = new URL(stringUrl);
                HttpURLConnection connection = (HttpURLConnection) url.openConnection();
                lectura = connection.getInputStream();
                escritura = new FileOutputStream(ficheroDescargado.getPath());
                byte[] buffer = new byte[1024];
                int count;
                while ((count = lectura.read(buffer)) > 0) {escritura.write(buffer, 0,
count);}
                escritura.flush();
                result = Activity.RESULT_OK;
            } catch (Exception e) {e.printStackTrace();}
            finally {
                if (escritura != null) {try {escritura.close();} catch (IOException e)
{e.printStackTrace();}}
                if (lectura != null) {try {lectura.close();} catch (IOException e)
{e.printStackTrace();}}
            }

            // Cuando se ha terminado la tarea de descarga lanzamos el receiver
            Intent intentResultado = new Intent();
            intentResultado.setAction("descargaCompletada");
            intentResultado.putExtra("rutaFichero", Environment.DIRECTORY_DOWNLOADS +
"/" + nombreFichero);
            intentResultado.putExtra("result", result);
            LocalBroadcastManager.getInstance(this).sendBroadcast(intentResultado);
        }
    }
}
```

El método onHandleIntent primero se encarga de la tarea de escribir el fichero en la carpeta indicada

El método onHandleIntent al final envía el anuncio

Código ReceiverFinDescarga

```
public class ReceiverFinDescarga extends BroadcastReceiver {
    public ReceiverFinDescarga() {
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        Bundle bundle = intent.getExtras();
        if (bundle != null) {
            String donde = bundle.getString("rutaFichero");
            int resultCode = bundle.getInt("result");
            if (resultCode == Activity.RESULT_OK) {
                Toast.makeText(context, "Descarga guardada en: " + donde,
Toast.LENGTH_LONG).show();
            } else {
                Toast.makeText(context, "Fallo en descarga", Toast.LENGTH_LONG).show();
            }
        }
    }
}
```

Práctica propuesta

Realiza la práctica propuesta en Ejercicios_18