

**MP_0489. Programación
multimedia y dispositivos móviles**

**UF2. Programación de
aplicaciones para dispositivos móviles**

2.6. Servicios

Índice

☰	Objetivos	3
☰	Qué es un servicio	4
☰	Tipos de clases en los servicios	7
☰	Funcionamiento de un servicio	10
☰	Ejemplo	15
☰	Resumen	24

Objetivos

Con esta unidad perseguimos los siguientes objetivos:

1

Conocer la clase *Service* en Android.

2

Entender las distintas fases del ciclo de vida de la clase *Service*.

3

Descubrir los distintos tipos de *Service*.

¡Ánimo y adelante!

Qué es un servicio

Un *Service* o servicio es un componente de una aplicación que realiza operaciones de larga ejecución en segundo plano.

De este modo, la aplicación puede iniciar un servicio y continuar ejecutándose en segundo plano, aunque el usuario cambie a otra aplicación.

La misión de los servicios en Android es ejecutar operaciones de larga duración. Para ello, y en oposición a la misión de una *Activity*, **un *Service* no depende de una interfaz de usuario**.

Operaciones y *services*

Pero, ¿qué operaciones de larga duración puede interesarnos realizar en un dispositivo móvil? Pensemos, por ejemplo, en una aplicación que permite acceder, gestionar y compartir ficheros en nuestra nube privada. Esta aplicación ejecutaría tareas que requieran descargar o subir ficheros a un servidor remoto.

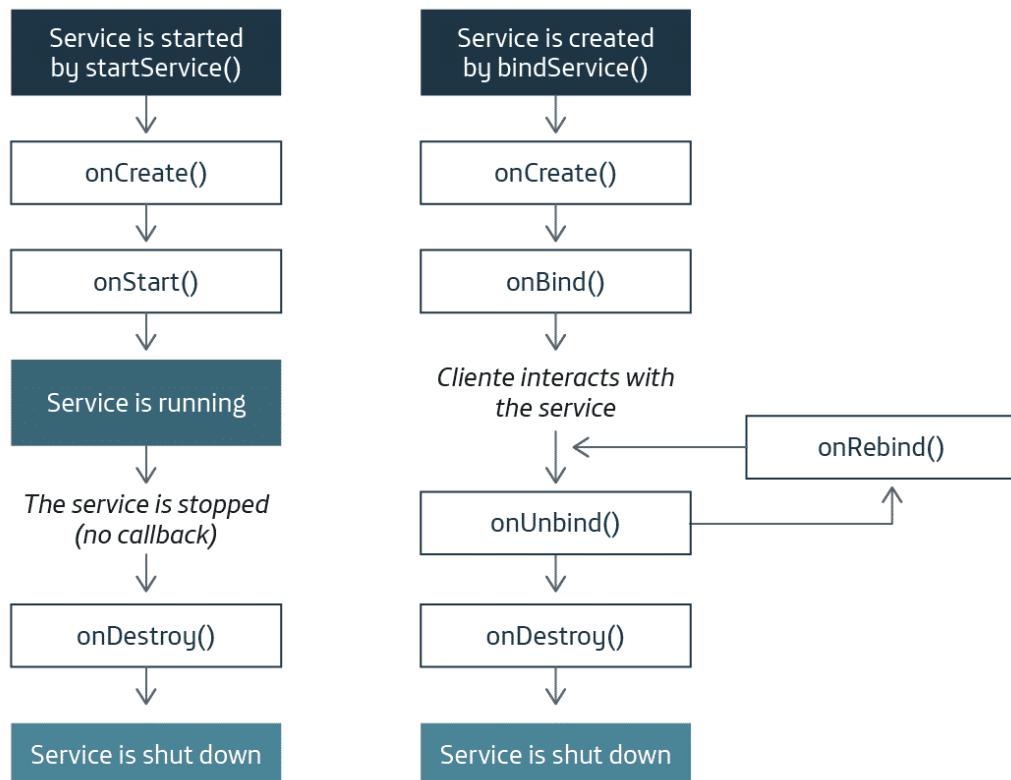
La clase *Activity* no es apropiada, ya que está concebida para proporcionar una interfaz de usuario a la que está íntimamente ligada. Una instancia de *Activity*, de forma general, debería interrumpir o pausar cualquier trabajo que ejecute en segundo plano cuando el usuario la retire de la pantalla.

Pero contamos con un *Service* para ayudarnos a implementar este tipo de trabajos.

La diferencia fundamental entre la *Activity* y el *Service* está en su ciclo de vida.

Cuando queremos desarrollar un servicio, debemos escribir una clase que extienda a la clase *Service* o alguna de sus herederas. Al hacerlo, tendremos que sobrecargar algunos métodos específicos de *Service*, que serán llamados en nuestra aplicación en respuesta a las peticiones de los componentes que actúen como clientes del servicio.

En dichos métodos, tendremos que iniciar las tareas a ejecutar. El *framework* se compromete, por medio de la especificación del ciclo de vida del *Service*, a no interrumpir la operación del servicio, aunque los clientes que inicien operaciones en él se retiren.



Ciclo de vida de un *service*.

A pesar de que su misión es la ejecución de tareas en segundo plano, **la gran mayoría de los métodos de entrada en los servicios son ejecutados por Android en el hilo principal de la aplicación.**



Es nuestra responsabilidad garantizar que **las tareas se ejecutarán en otros hilos para no ocupar durante demasiado tiempo el hilo principal.**

Tipos de servicios

Disponemos de tres tipos de servicios:

1

Started Service

Soportan la misión fundamental de la clase *Service*; solo en este modo de funcionamiento **el servicio permanecerá activo indefinidamente hasta que él mismo indique que ha terminado su trabajo, o hasta que lo paremos explícitamente.**

2

Bound Service

Ofrecen la posibilidad de que **uno o varios componentes obtengan una referencia a un objeto con el que interactuar de forma directa con el servicio, en lugar de con peticiones asíncronas lanzadas vía *Intents*.**

3

Scheduled Service

Fueron incorporados con Android 5 (nivel de API 21), y Google recomienda expresamente su uso frente a los otros dos para todas las versiones de Android que los soportan. Su utilización supone un cambio drástico para las aplicaciones, ya que los servicios programados no se ejecutan inmediatamente después de las peticiones de sus clientes. Una petición de un trabajo programado supone que el cliente defina algunas condiciones que deben ser ciertas para que el trabajo sea realizado por el servicio.

Tipos de clases en los servicios

En el siguiente apartado te mostraremos los tipos de clases en los servicios y cómo puedes declarar e iniciar un servicio.

Existen dos tipos de clases en los servicios: la clase **Service** y la clase **IntentService**.

1

Service

- Es un componente de la aplicación sin interfaz.
- Como componente de la aplicación, se ejecuta en el hilo principal.
- Para usarlo en segundo plano, hay que crear un nuevo hilo dentro del servicio.
- Puede vincularse (*bind*) a una *Activity*.
- Los *services* se inician y se paran mediante *Intents*, ya sean explícitos o de acción.

2

IntentService

- Es un tipo especial de servicio que se ejecuta en segundo plano.
- Está pensado para realizar una tarea que requiera un largo procesamiento en segundo plano y después pararse solo.
- Su funcionamiento está desaconsejado para realizar tareas que tengan que ejecutarse indefinidamente.

Cómo declarar un servicio

Los servicios **deben aparecer en el AndroidManifest.XML**. Deben declararse dentro del TAG *application*, independientemente del tipo de servicio que sea.

```
<application
    android:name="@string/app_name"
    android:theme="@android:style/Holo">
    <service name="com.myapp.id.MyService" label="@string/service_label">
        <intent-filter>
            <action android:name = "com.myapp.id.action.ACTION_OPEN_SERVICE"/>
        </intent-filter>
    </service>
</application>
```

Es importante destacar que el parámetro “name” debe ser la ruta en la que se encuentra el servicio dentro de la carpeta *src* de nuestro proyecto. Además, hemos establecido un *intent-filter*, de manera que podemos iniciar el servicio de dos formas.

Cómo iniciar un servicio

Existen dos métodos diferentes:

1. Mediante un *intent* explícito:

```
Intent intent = new Intent(this, MyService.class);
startService(intent);
```

2. Mediante un *intent* de acción:

```
Intent intent = new Intent("com.myapp.id.action.ACTION_OPEN_SERVICE");
startService(intent);
```

En el siguiente apartado aprenderemos cómo funcionan los *services*.

Funcionamiento de un servicio

Ya hemos visto en qué consiste y cuáles son sus principales tipos y clases. Pero, ¿cómo funciona realmente un *service*?

Lo aprenderemos mediante un ejemplo, que extiende de *Service* con los cuatro métodos principales.

```
public class MyService extends Service{
    @Override
    public void onCreate() {
        super.onCreate();
        Log.i(getClass().getSimpleName(), "Creating service");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        super.onStartCommand(intent, flags, startId);
        Log.i(getClass().getSimpleName(), "Intent received");
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        App.getInstance().cancelPendingRequests(getClass().getSimpleName());
        super.onDestroy();
    }

    @Override
    public IBinder onBind(Intent intent) {
        return new IBinder(this);
    }
}
```

Veamos ahora esos métodos:

1

onCreate()

De manera similar a una *Activity*, se ejecuta la primera vez que se crea el servicio y sirve para inicializar variables.

2

onStartCommand()

Se ejecuta cuando el servicio recibe un *Intent* lanzado mediante el comando *startService*. Si el servicio ya está iniciado, los *intents* sucesivos no volverán a generar un *onCreate*, si no que pasarán directamente aquí.

3

onBind()

Este comando es llamado cuando se ejecuta un *bindService()* desde una *Activity*.

4

onDestroy()

Cuando se llama al comando *stopService()*, o dentro del mismo *Service* al método *stopSelf()*, se pasa por este método, en el que deben terminarse hilos o tareas que puedan estar ejecutándose.

Como ya hemos indicado, todo este código funcionaría en primer plano. Si deseamos crear un hilo para realizar tareas en segundo plano, habría que hacerlo en *onStartCommand*.

```
public class MyService extends Service{
    @Override
    public void onCreate() {
        super.onCreate();
        Log.i(getClass().getSimpleName(), "Creating service");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        super.onStartCommand(intent, flags, startId);
        if(workerThread == null || !workerThread.isAlive()){
            workerThread = new Thread(new Runnable(){
                public void run(){
                    Log.i(getClass().getSimpleName(), "Corriendo en otro hilo");
                }
            });
            workerThread.start();
        }
        return START_STICKY;
    }

    @Override
```

```
public void onDestroy() {
    App.getInstance().cancelPendingRequests(getClass().getSimpleName());
    super.onDestroy();
}

@Override
public IBinder onBind(Intent intent) {
    return new IBinder(this);
}
```

Hay que destacar que **este tipo de servicios hay que pararlos explícitamente, aunque el sistema también podría hacerlo si se encontrase escaso de memoria.** Si en el método *onStartCommand* se ha devuelto START_STICKY, el sistema podrá revivir el servicio cuando vuelva a tener memoria disponible.

Por otro lado, un *IntentService* es mucho más sencillo, ya que funciona directamente en segundo plano, y está pensado para responder a un *Intent* y realizar el trabajo que se le ha indicado.

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

    @Override
    protected void onHandleIntent(Intent intent) {
        //Mostrar notificaciones o cualquier otra cosa que se quiera hacer;
    }
}
```

En este tipo de servicio, cuando se ejecute un *startService(intent)* se llamará al método *onHandleIntent*, se ejecutará en un hilo secundario todo el trabajo y se parará el servicio automáticamente.

Tipos de servicios según el modo de creación

Como hemos visto, existen dos tipos de servicios dependiendo del modo en que hayan sido creados. **Las funciones de estos servicios son diferentes, y por lo tanto, también su ciclo de vida.**



Si el servicio es iniciado mediante `startService()` el sistema comenzará creándolo y llamando a su método `onCreate()`. A continuación, llamará a su método `onStartCommand()` con los argumentos proporcionados por el cliente. El servicio continuará en ejecución hasta que sea invocado el método `stopService()` o `stopSelf()`.

Aunque se produzcan varias llamadas a `startService()` no supondrá la creación de varios servicios, pero sí 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()` 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**.
- Devolveremos **START_NOT_STICKY** si queremos que el servicio sea creado de nuevo solo cuando llegue una nueva solicitud de creación.

 También podemos utilizar `bindService()` para obtener una conexión persistente con un servicio.

También podemos utilizar `bindService()` para obtener una conexión persistente con un servicio.

Si dicho servicio no está en ejecución, será creado llamando al método `onCreate()`, pero no se llamará a `onStartCommand()`. En su lugar se llamará al método `onBind()`, que devolverá 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 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.

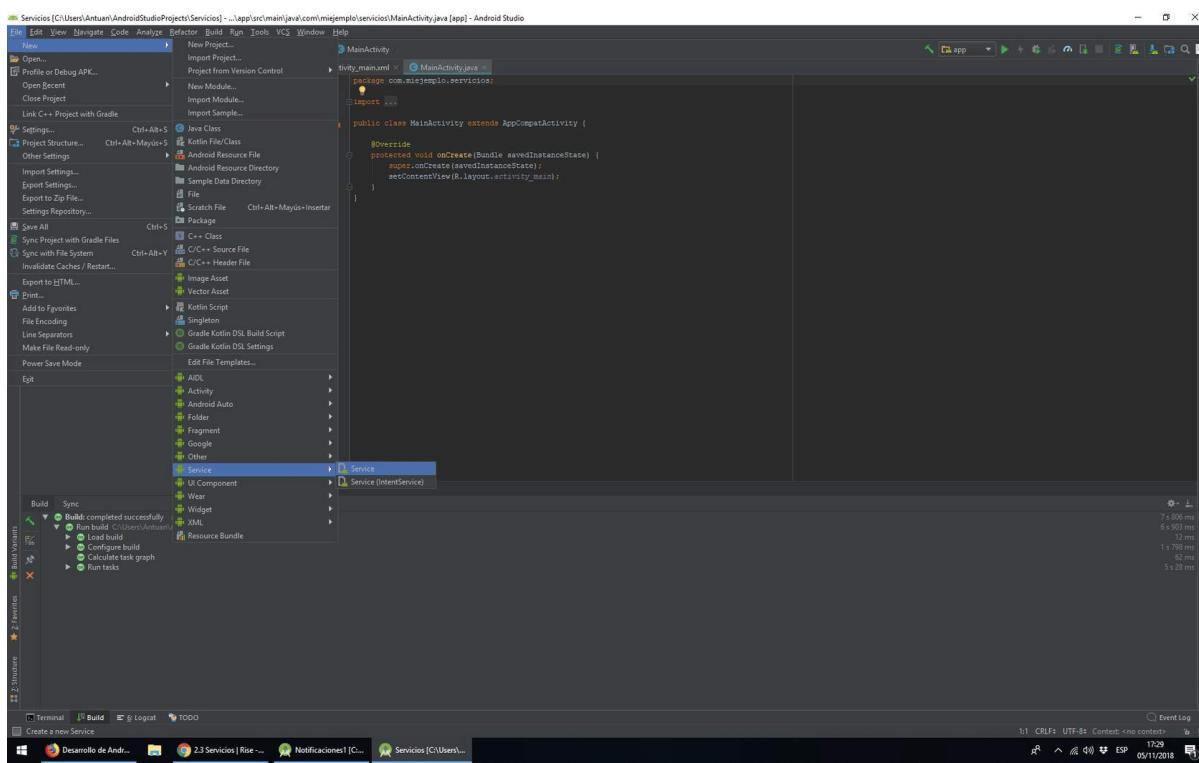
Ejemplo

Veamos ahora un ejemplo de cómo crear un servicio con Android Studio.

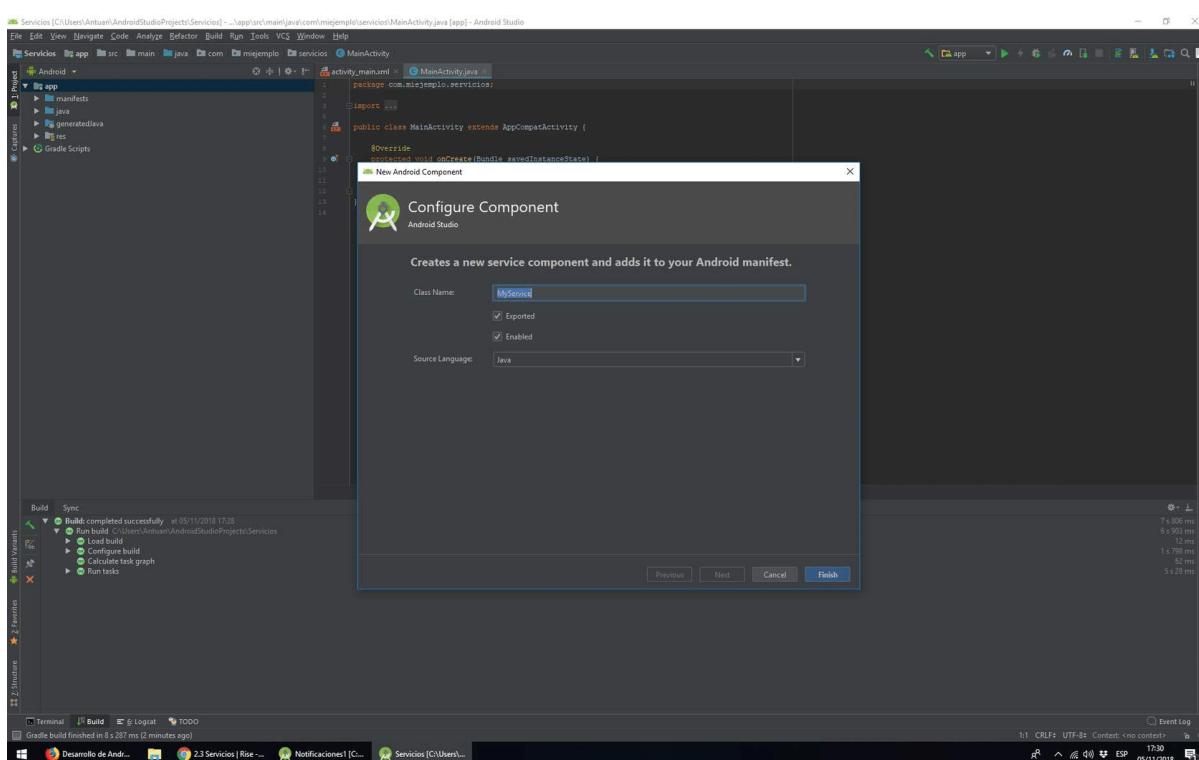
Crear un nuevo servicio

Para crear un nuevo servicio simplemente tendremos que hacer clic en New -> Service -> Service.

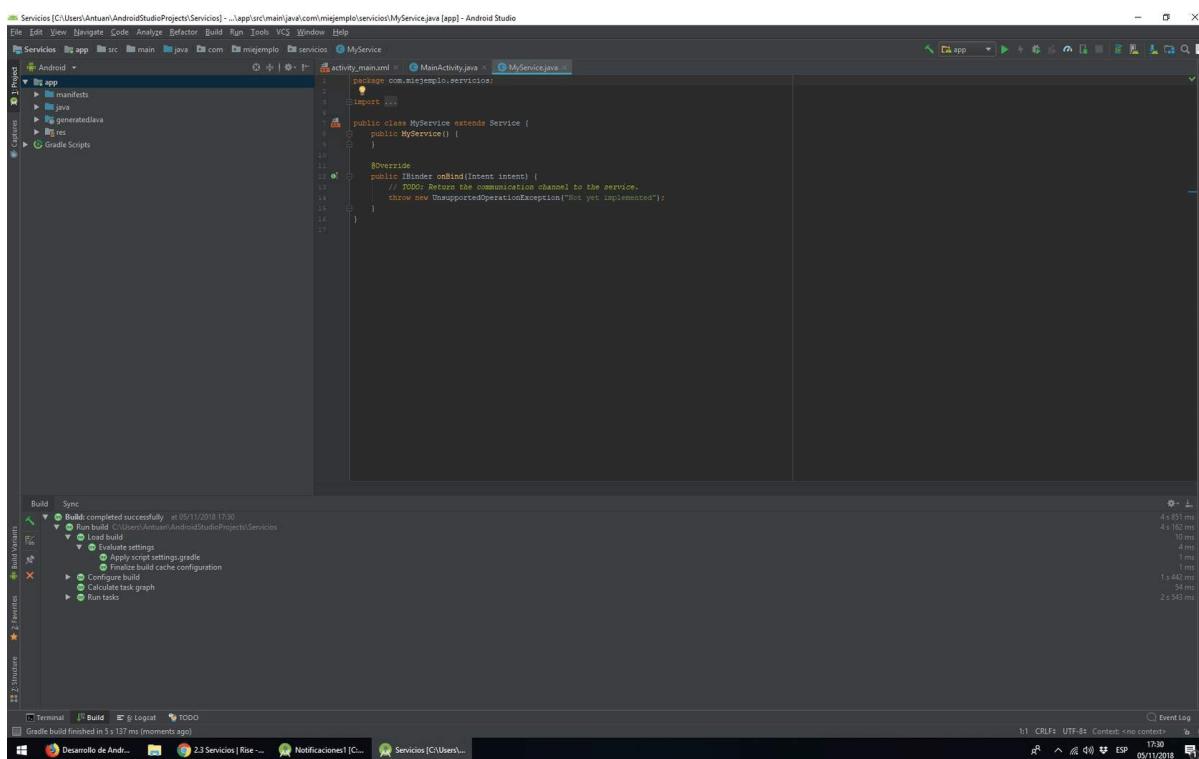
Paso 1



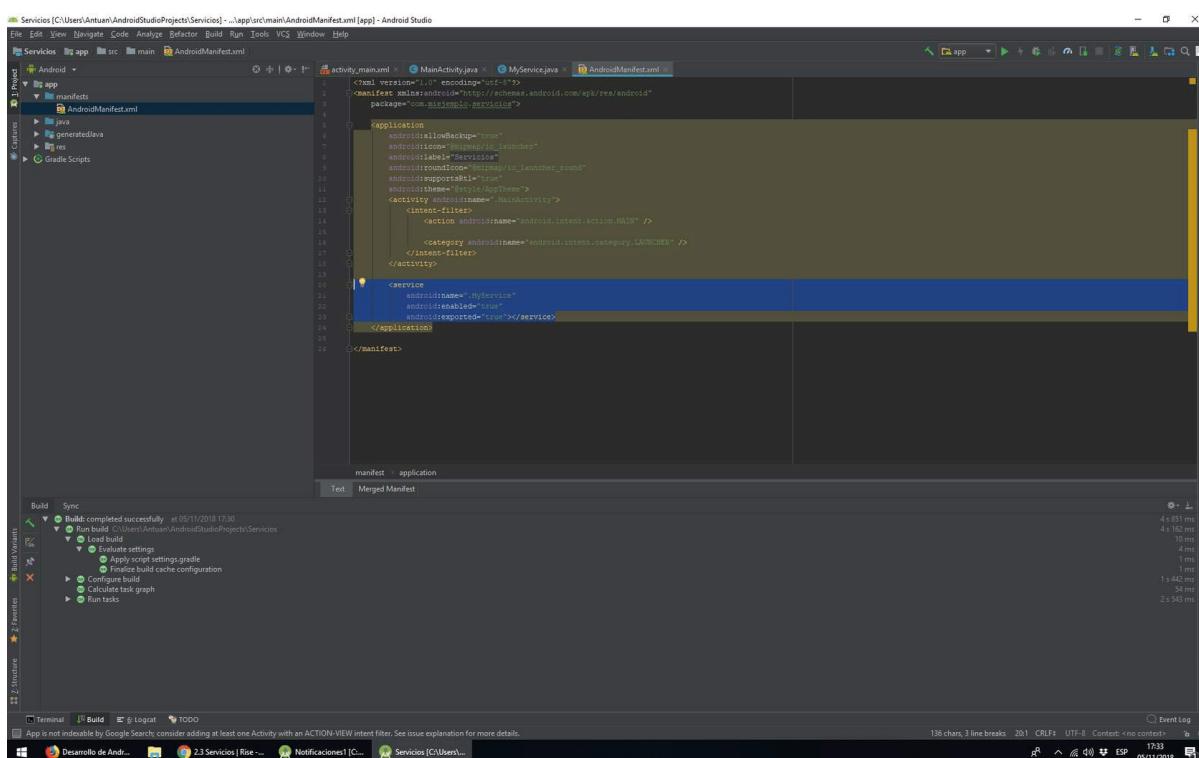
Paso 2



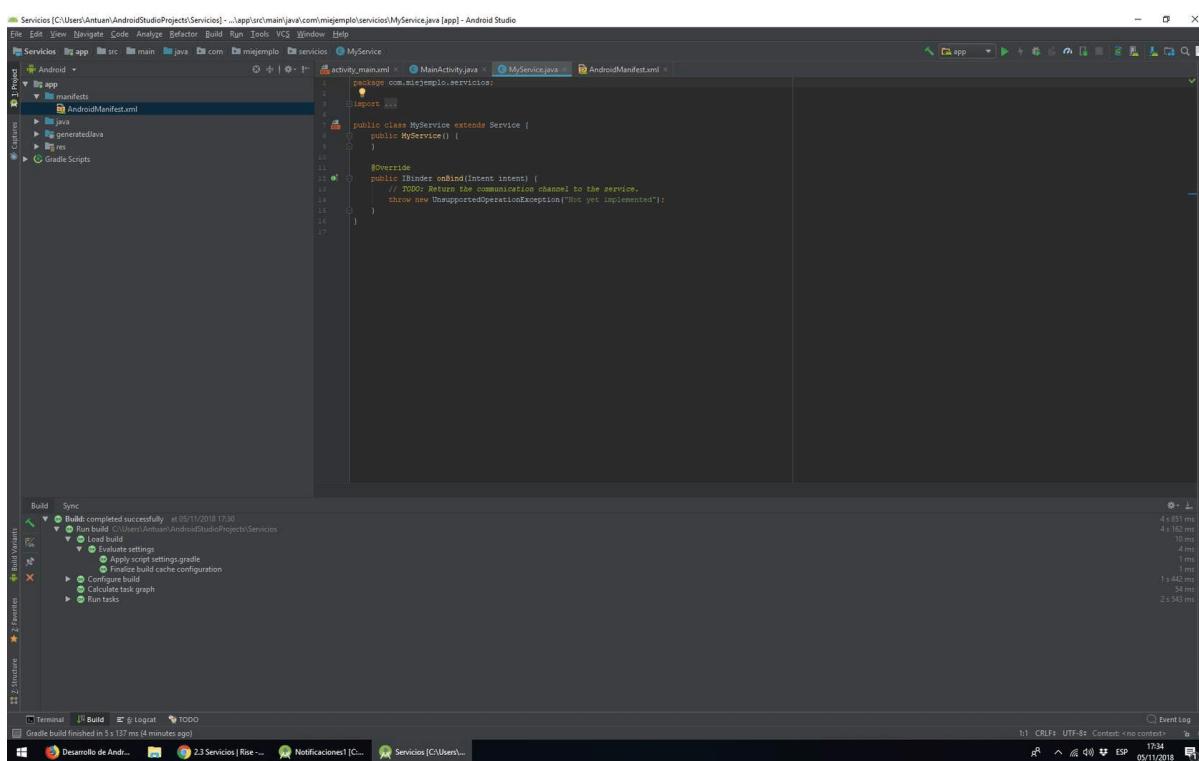
Paso 3



Paso 4



Paso 5



¡Hecho!

De esta forma, el servicio se declarará de forma automática en el *manifest*, y se creará automáticamente la clase que hereda de *Service*.

Veamos un ejemplo práctico:

Vamos a realizar una aplicación sencilla que nos permita **activar y desactivar un servicio**. Haremos que nos muestre la hora actual cada 5 segundos en una *Toast*.

1

Agregamos un par de **botones** con los que podamos activar y desactivar el servicio en el *layout* de la actividad principal.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="-
    match_parent"
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Descripción"
        android:id="@+id/textView"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="40dp" />

    <Button
        android:layout_width="100dp"
        android:layout_height="50dp"
        android:text="Encender"
        android:id="@+id/buttonOn"
```

```
        android:layout_marginTop="50dp"
        android:layout_below="@+id/textView"
        android:layout_alignLeft="@+id/buttonOff"
        android:layout_alignStart="@+id/buttonOff" />

    <Button
        android:layout_width="100dp"
        android:layout_height="50dp"
        android:text="Apagar"
        android:id="@+id/buttonOff"
        android:layout_below="@+id/buttonOn"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="50dp" />

</RelativeLayout>
```

2

A continuación, desde la actividad principal, iniciaremos o detendremos el servicio, dependiendo de si pulsamos el botón *Encender* o *Apagar*.

```
package com.miejemplo.servicios;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

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

        findViewById(R.id.buttonOn).setOnClickListener(mClickListener);
        findViewById(R.id.buttonOff).setOnClickListener(mClickListener);
    }

    View.OnClickListener mClickListener = new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            switch (v.getId()){
                case R.id.buttonOn:
                    // Start Service
                    startService(new Intent(MainActivity.this, MyService.class));
                    break;
                case R.id.buttonOff:
                    // Stop Service
                    stopService(new Intent(MainActivity.this, MyService.class));
                    break;
            }
        }
    };
}
```

3

Para terminar, definimos un **AsyncTask** en el servicio que hemos creado. Desde aquí obtendremos la hora actual cada 5 segundos y la mostraremos con una notificación de tipo **Toast**.

En el método **onStartCommand()** se ejecutará el **AsyncTask** mientras que se cancelará en el **onDestroy()**.

```
package com.miejemplo.servicios;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;
import android.os.AsyncTask;
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;

public class MyService extends Service {

    MyTask myTask;

    @Override
    public void onCreate() {
        super.onCreate();
        Toast.makeText(this, "Servicio creado!", Toast.LENGTH_SHORT).show();
        myTask = new MyTask();
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        myTask.execute();
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Servicio destruido!", Toast.LENGTH_SHORT).show();
        myTask.cancel(true);
    }

    @Override
    public IBinder onBind(Intent intent) {
        throw new UnsupportedOperationException("Not yet implemented");
    }

    private class MyTask extends AsyncTask<String, String, String> {

        private DateFormat dateFormat;
        private String date;
        private boolean cent;

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
            dateFormat = new SimpleDateFormat("HH:mm:ss");
            cent = true;
        }

        @Override
        protected String doInBackground(String... params) {
            while (cent){
```

```
date = dateFormat.format(new Date());
try {
    publishProgress(date);
    // Stop 5s
    Thread.sleep(5000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
}
return null;
}

@Override
protected void onProgressUpdate(String... values) {
    Toast.makeText(getApplicationContext(), "Hora actual: " + values[0],
Toast.LENGTH_SHORT).show();
}

@Override
protected void onCancelled() {
    super.onCancelled();
    cent = false;
}
}
```

Si pulsamos el botón *Encender*, observaremos que se muestra la hora actual aunque salgamos de la aplicación. No parará de ejecutarse hasta que volvamos a la aplicación y pulsemos el botón *Desactivar el servicio*.

Así sería el resultado:







Resumen

Hemos terminado la lección, repasemos los puntos más importantes que hemos tratado.

- A lo largo de esta unidad hemos conocido la **clase Service** y su importancia para ejecutar tareas largas en segundo plano. También hemos analizado los tres tipos de **services** de que disponemos.
- Hemos visto cómo **declarar, iniciar y ejecutar un Service**.
- Hemos aprendido los distintos **métodos de la clase Service**.
- Y, para finalizar, hemos creado un **ejemplo de uso de Service**.



PROEDUCA