Servicios avanzados - Ejercicios

Índice

1. Servicio reproductor de música

Vamos a crear un servicio que inicie la reproducción de un recurso audio al arrancarse, y que detenga la reproducción al pararse.

- Descargad las plantillas de la sesión. En el proyecto `AJDM_S16_1` tenemos una actividad principal que muestra un botón Start y un botón Stop. En sus respectivos `OnClickListener`'s tendremos que iniciar y parar el servicio con los métodos `startService(...)` y `stopService(...)`, pasándoles en ambos casos un `new Intent(main, MiAudioServicio.class)` como parámetro. Pero para ello tendremos que crear antes la clase que define el servicio:
- Creamos una nueva clase Java que se llame `MiAudioServicio` y sobrecargamos los métodos `onStartCommand`, `onCreate`, `onDestroy` y `onBind`, ayudándonos de las herramientas que proporciona Eclipse.
- Declaramos un campo `private MediaPlayer mediaPlayer;` en la clase del servicio.
- Cuando iniciemos el servicio desde la actividad, primero se creará y se invocará al método `onCreate(...)`. En él crearemos el reproductor:

```
Toast.makeText(this,"Servicio creado ...",
Toast.LENGTH_LONG).show();
mediaPlayer = MediaPlayer.create(getApplicationContext(),
R.raw.ubuntu);
mediaPlayer.setLooping(true);
```

  mostrando un `Toast` para quedarnos tranquilos de que el servicio se ha iniciado. El recurso R.raw.ubuntu es un archivo .ogg que se incluye en la carpeta `res/raw` de las plantillas del proyecto. También podía haber sido un mp3.
- Una vez creado, se ejecutará el método `onStartCommand(...)`. En él iniciaremos la reproducción y devolveremos el valor `Service.START_STICKY`.

```
mediaPlayer.start();
return Service.START_STICKY;
```

- Finalmente, al destruir el servicio, detendremos la reproducción y mostraremos un `Toast`:

```
Toast.makeText(this,"onDestroy: Servicio destruido.",
Toast.LENGTH_LONG).show();
mediaPlayer.stop();
```

- En cuanto al método `onBind`, devolveremos `null`, que indica que el servicio no tiene definido un interfaz AIDL para comunicarse con otros.
- Para que el servicio funcione en la aplicación, habrá que declararlo en el `AndroidManifest.xml`, dentro de `application`:

```
                ...
                <service android:enabled="true"
android:name=".MiAudioServicio"/>
    </application>
```

Si todo ha ido bien, y si hemos implementado los listeners de los botones que inician y detienen el servicio, debería funcionar. Probad iniciar el servicio y salir de la aplicación, entrar en otras, etc. El sonido seguirá reproduciéndose. Para detenerlo, volvemos a abrir la aplicación y lo detenemos.

2. Servicio con proceso en background. Contador

Los servicios se utilizan para ejecutar algún tipo de procesamiento en background. En el anterior ejercicio utilizamos el reproductor del sistema y simplemente le indicamos cuándo iniciarse y cuándo detenerse. En este ejercicio vamos a crear nuestro propio proceso que ejecute determinada tarea, en este caso, que vaya contando desde 1 hasta 100, deteniéndose 5 segundos antes de cada incremento. En cada incremento mostraremos un `Toast` que nos informe de la cuenta.

En las plantillas tenemos el proyecto `AJDM_S16_2` que ya incluye la declaración del servicio en el manifest, la actividad que inicia y detiene el servicio, y el esqueleto del servicio `MiCuentaServicio`.

- En el esqueleto que se proporciona, viene definida una extensión de `AsyncTask` llamada `MiTarea`. Podía no haberse incluido, ya que sólo tendríais que crear una nueva clase Java que herede de `AsyncTask`, y sobrecargar con la ayuda de Eclipse los métodos `onPreExecute`, `doInBackground`, `onProgressUpdate` y `onCancelled`. Se pide implementarlos, el primero de ellos inicializando el campo `i` que se utiliza para la cuenta, el segundo ejecutando un bucle desde 1 hasta 100, y en cada iteración pidiendo mostrar el progreso y durmiento después 5 segundos con `Thread.sleep(5000)`. El tercer método, `onProgressUpdate` mostrará el `Toast` con el progreso, y por último el método de cancelación pondrá el valor máximo de la cuenta para que se salga del bucle.
- En los métodos del servicio, `onCreate`, `onStartCommand` y `onDestroy`, introduciremos la creacion de la nueva `MiTarea`, su ejecución (método `execute()` de la tarea) y la cancelación de su ejecución (método `cancel()` de la tarea).

Una vez más, el servicio deberá seguir funcionando aunque se salga de la aplicación y podrá ser parado entrando de nuevo en la aplicación y pulsando Stop.

3. Servicio con notificaciones. Números primos

Este ejercicio es una extensión del anterior, pero vamos a utilizar un nuevo proyecto plantilla, el `AJDM_S16_3`. En lugar de mostrar cualquier número de la cuenta, vamos a mostrarlos sólo si son primos. Además, en lugar de mostrar un `Toast`, vamos a mostrar una `Notification` que aparecerá en la barra de tareas y se actualizará con la llegada de cada nuevo número. Si salimos de la aplicación sin parar el servicio, seguirán apareciendo notificaciones, y si pulsamos sobre la notificación, volverá a lanzar la actividad, cerrándose la notificación que hemos pulsado.

- Dentro del servicio `MiNumerosPrimosServicio` se encuentra declarada la `AsyncTask` llamada `MiTarea`. En ella tenemos como campos de la clase una `Notification` y un `NotificationManager`. Hay que darles valores en el método `onPreExecute()`.
- El método `doInBackground(...)` ejecutará un bucle que irá incrementando `i` mientras su valor sea menor de `MAXCOUNT`. En cada iteración, si el número es primo (función incluida en la plantilla), pedirá que se muestre el progreso, pasándole como parámetro el nuevo primo encontrado.
- Implementar el método `onProgressUpdate(...)` para que muestre la notificación. Para ello habrá que actualizar la notificación con el método `setLatestEventInfo`, al cuál le pasaremos en un String la información del último primo descubierto y le pasaremos un `PendingIntent` para que al pulsar sobre la notificación, nos devuelva a la actividad de la aplicación, por si la hemos cerrado. Para crear el PendingIntent utilizaremos el método `PendingIntent.getActivity(...)` al cuál le tenemos que pasar un `new Intent(getApplicationContext(),Main.class)`.
- La aplicación debería funcionar en este punto, mostrando las notificaciones y relanzando la aplicación si son pulsadas, pero no cerrándolas al pulsarlas. Para ello simplemente tenemos que llamar al método `cancel(id)` del `notificationManager` y pasarle la constante NOTIF_ID para que la notificación no se muestre como una nueva, sino como actualización de la que ya habíamos puesto. Una manera de hacerlo es en un método estático del `MiNumerosPrimosServicio`, que podemos llamar `cerrarMiNotificacion(NotificationManager nm)`. Este método será invocado desde el `Main.onResume()`.

Notificación del servicio de números primos

4. IP AppWidget

En programación de Android se denomina `Widget` a los componentes de alto nivel de la interfaz de usuario, y `AppWidgets` a los widgets que se pueden añadir al escritorio del sistema operativo, como el reloj, pequeños controles, etc.

Vamos crear un proyecto `AJDM_S16_4` para construir un AppWidget de Android, que nos muestre en todo momento la IP que el dispositivo está usando en este momento. No necesitaremos ninguna actividad, así que podemos desmarcar la casilla "Create activity", o bien eliminar la actividad después (no sólo la clase, sino también la declaración en el manifest).

En el proyecto pulsamos con el boton derecho y añadimos un nuevo Android XML File, de tipo AppWidget Provider, que se llame `miwidget.xml`. El editor nos permite pulsar sobre el AppWidget Provider y editar sus atributos. Ponemos los siguientes:

```
android:minWidth="146dip"
android:minHeight="72dip"
android:updatePeriodMillis="600000"
android:initialLayout="@layout/miwidget_layout"
```

El miwidget_layout lo tenemos que crear, o dará error. Así que creamos un nuevo Android XML File de tipo Layout llamado `miwidget_layout.xml` y le añadimos un

campo de texto `TextView` con el texto vacío.

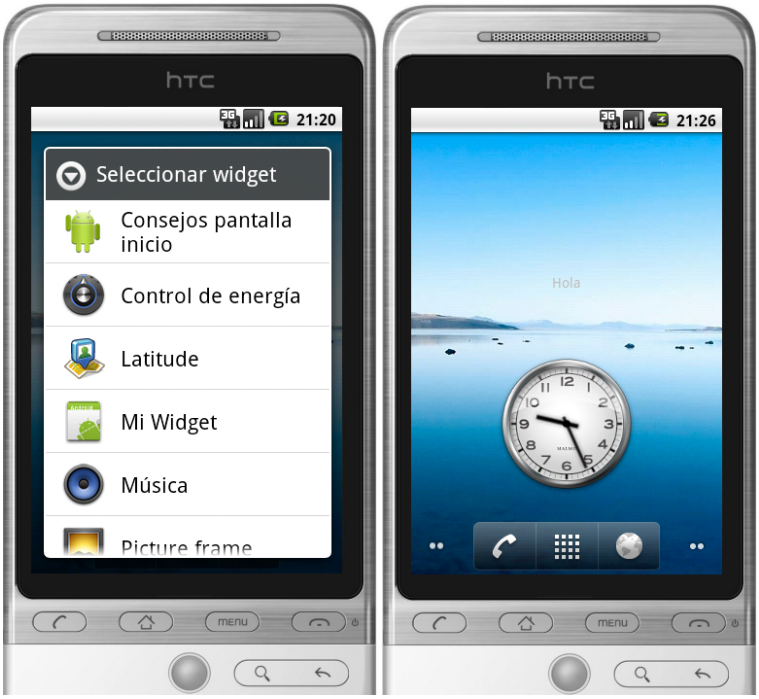
Creamos una clase `MiWidget` que herede de `AppWidgetProvider`, en el paquete `es.ua.jtech.ajdm.appwidget`. Sobrecargamos su método `onUpdate(...)` y actualizamos en él el campo de texto, usando `RemoteViews` y pasándoselos al `AppWidgetManager`:

```
RemoteViews updateViews = new RemoteViews(context.getPackageName(),
R.layout.miwidget_layout);
updateViews.setTextViewText(R.id.TextView01,"Hola");
ComponentName thisWidget = new ComponentName(context,
MiWidget.class);
AppWidgetManager.getInstance(context).updateAppWidget(thisWidget,
updateViews);
```

Antes de probar el widget hay que declararlo en el `AndroidManifest.xml`, dentro de `application`:

```
        <receiver android:name=".MiWidget" android:label="Mi
Widget">
            <intent-filter>
                <action
android:name="android.appwidget.action.APPWIDGET_UPDATE" />
            </intent-filter>
            <meta-data android:name="android.appwidget.provider"
                       android:resource="@xml/miwidget" />
        </receiver>
```

Ejecutamos el widget desde Eclipse, como aplicación android, y comprobamos que no ocurra ningún error en la consola de Eclipse. Ya se puede añadir el widget en el escritorio, efectuando una pulsación larga sobre una porción de área libre del escritorio, y seleccionando nuestro widget.



Instalación del AppWidget en el emulador

Si todo funciona correctamente, vamos a implementar en el `MiWidget` un servicio `UpdateService` que realizará la actualización del widget, evitando así bloqueos debidos a la velocidad de la red. El servicio recogerá la información que le devuelve en texto plano la página http://www.whatismyip.org y la mostrará en el campo de texto del widget.

Instrucciones para programar el servicio que se pide:

- Creamos la clase `public static class UpdateService extends Service` dentro de la clase `MiWidget` y sobrecargamos los métodos `onBind` (que es obligatorio, pero devolverá `null`) y `onStartCommand` que devolverá `Service.START_STICKY`.
- Hay que declarar el servicio en el `AndroidManifest.xml`, dentro de `application`, con:

```
<service android:name=".MiWidget$UpdateService" />
```

- Del método `MiWidget.onUpdate(...)` podemos cortar todas las líneas y sustituirlas por la llamada al servicio:

```
        context.startService(new Intent(context,
UpdateService.class));
```

- En el método `onStartCommand` del servicio, pegaremos las líneas que actualizan los RemoteViews, pero las modificaremos para que obtengan el contexto y el paquete del widget, quedando el método así:

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
        RemoteViews updateViews = new RemoteViews(getPackageName(),
                        R.layout.miwidget_layout);
        updateViews.setTextViewText(R.id.TextView01,"Hola Serv");
        ComponentName thisWidget = new ComponentName(this,
MiWidget.class);
AppWidgetManager.getInstance(this).updateAppWidget(thisWidget,
                        updateViews);
        return Service.START_STICKY;
}
```

Ahora podemos volver a probar el widget, ejecutándolo desde Eclipse. Si funciona, podemos pasar a sustituir la línea

```
updateViews.setTextViewText(R.id.TextView01,"Hola Serv");
```

por el código que accede a la URL por HTTP, obteniendo un InputStream y convirtiendo los bytes a String para mostrarlo:

```
                    String ipstring = "Unknown IP";

            try {
                            URL url = new
URL("http://www.whatismyip.org");
                            HttpURLConnection http =
(HttpURLConnection)url.openConnection();
                            InputStream is = http.getInputStream();
                            byte[] buffer = new byte[20];
                            is.read(buffer, 0, 20);
                            ipstring = "IP: "+new String(buffer);
            } catch (MalformedURLException e) {
                            e.printStackTrace();
                    } catch (IOException e) {
                            e.printStackTrace();
                    }

updateViews.setTextViewText(R.id.TextView01,ipstring);
```

Antes de probarlo hay que añadir el permiso de Internet en el `AndroidManifest.xml`, fuera de `application`:

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

Ejecutamos y observamos el resultado:

Widget que muestra la IP

Se puede añadir un comportamiento al pulsar sobre algún componente del widget. Por ejemplo, para que se abra un navegador con la web consultada, añadiríamos las siguientes líneas para actualizar el `updateViews`:

```
        Intent defineIntent = new Intent(Intent.ACTION_VIEW,
                Uri.parse("www.whatismyip.org"));
         PendingIntent pendingIntent = PendingIntent.getActivity(
                getApplicationContext(), 0, defineIntent, 0);
        updateViews.setOnClickPendingIntent(R.id.miwidgetlayout,
pendingIntent);
```

Nota:

Para que la referencia al recurso `R.id.miwidgetlayout` funcione, se tiene que definir el atributo `android:id="@+id/miwidgetlayout"` del `LinearLayout` del widget, que se encuentra en el archivo `miwidget_layout.xml`.