UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA POLITÉCNICA SUPERIOR DE GANDÍA

Máster en Ingeniería Acústica

DESARROLLO DE UN PAQUETE DE SOFTWARE ACÚSTICO PARA DISPOSITIVOS MÓVILES BASADOS EN ANDROID

TRABAJO FINAL DE MÁSTER

Autor:
Gabriel Moreno Ibarra

Director/es:
D. Francisco Javier Redondo Pastor
D. Jordi Bataller Mascarell

GANDIA, 2012

DESARROLLO DE UN PAQUETE DE SOFTWARE ACÚSTICO PARA DISPOSITIVOS MÓVILES BASADOS EN ANDROID

Autor: Gabriel Moreno Ibarra
Director 1: D. Francisco Javier Redondo Pastor
Director 2: D. Jordi Bataller Mascarell

Resumen

En la actualidad, el uso de dispositivos móviles es un hábito muy cotidiano y en pleno auge. Existen varias plataformas: iOS, Symbian, Windows Phone, Blackberry… pero la que más se está extendiendo es, sin lugar a dudas, Android. En este trabajo se aplican los conocimientos acústicos aprendidos durante el Máster, para el desarrollo de una aplicación para este sistema operativo. En concreto, el software será una herramienta capaz de calcular el tiempo de reverberación en cualquier lugar y de forma gratuita.

Abstract

Nowadays, use of mobile devices is a daily habit and booming. There are several platforms: iOS, Symbian, Windows Phone, Blackberry... but the one that is spreading more is undoubtedly, Android. In this work the knowledge about acoustics learned during the Master is applied to develop an app for this operating system. Specifically, the software will be a tool to calculate the reverberation time anywhere for free.

Autor: Gabriel Moreno Ibarra, e-mail: gabmoib@epsg.upv.es
Fecha de entrega: 03-09-2012

ÍNDICE

I. OBJETO Y ANTECEDENTES

El objeto de este proyecto es la realización de una aplicación para smartphones con sistema operativo Android. Tiene que ser capaz de calcular el tiempo de reverberación en bandas de octava por medio de una medición sencilla y amigable para el usuario.

El rango de medición estará comprendido entre 125 Hz y 4 kHz. Por lo que se obtendrán resultados en las siguientes 6 bandas:

[125, 250, 500, 1000, 2000, 4000] (Hz)

Habrá que empezar por aprender el lenguaje/s que utiliza este SO[1] y familiarizarse con él, haciendo aplicaciones más sencillas hasta obtener un grado de entendimiento global suficiente como para poder abordar el problema en conjunto.

El hecho de que exista un mercado tan amplio, la creciente demanda de profesionales del sector, así como la posibilidad de emprender con ideas propias, son sin lugar a dudas, las tónicas desencadenantes para decidirse a la realización de este Trabajo Final de Máster.

Palabras clave: android, reverb, tiempo, app, cálculo.

[1] SO: Sistema Operativo.

II. INTRODUCCIÓN TEÓRICA

II.1. Tiempo de reverberación

La acústica de salas es la rama de la acústica que estudia la interacción del sonido con los elementos estructurales.

Desde hace miles de años existen pruebas acerca del conocimiento en el que la arquitectónica afecta a los fenómenos sonoros y cómo estos son percibidos por el ser humano. Pero no fue hasta casi finalizar el siglo XIX, cuando el físico Wallace Sabine (considerado el padre de la acústica moderna), dedujo que la calidad acústica de un recinto estaba íntimamente relacionada con la reverberación de éste. De hecho, incluso fue capaz de construir una ecuación para poder calcular el tiempo de reverberación teniendo en cuenta las dimensiones y la absorción de una sala.

Con el paso de los años, se ha ido añadiendo mayor precisión en la obtención de este parámetro.

Actualmente, uno de los métodos que se emplean para conseguir evaluar el RT es mediante la medición del tiempo de caída de una señal impulsiva. Básicamente, consiste en hacer una integración invertida en el tiempo de dicha señal elevada al cuadrado y extraer el tiempo que pasa desde que tenemos un determinado nivel de amplitud hasta otro (en esto se basa el algoritmo de la app de este trabajo).

El tiempo de reverberación es un parámetro que permite evaluar la calidad acústica de una sala. Dependiendo de su valor se puede caracterizar un recinto.

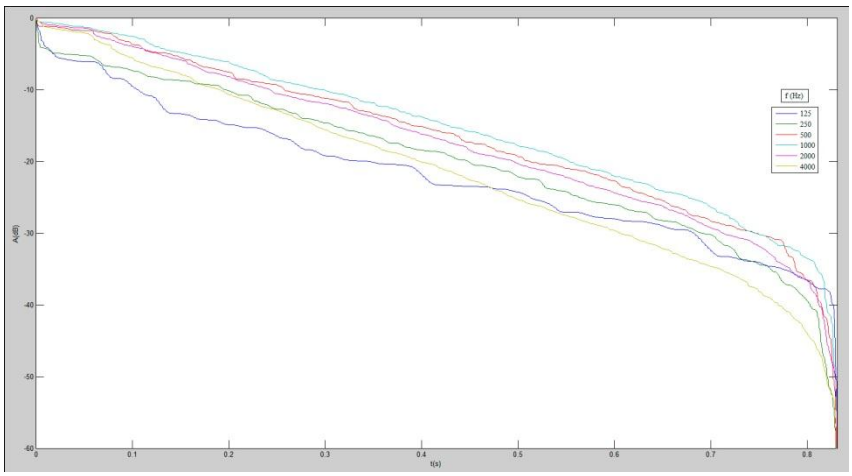

Fig. 1: Tiempo de reverberación.

Se define como el tiempo que tarda en decaer la energía de una fuente sonora desde que deja de emitir hasta que se ha reducido en 60 dB.

Existen otras maneras de medirlo, realizando una extrapolación:

- EDT[2]: (Caída entre 0 y -10 dB) x 6 [s]
- T20: (Caída entre -5 y -25 dB) x 3 [s]
- T30: (Caída entre -5 y -35 dB) x 2 [s]

Otro detalle a tener en cuenta es que conforme mayor sea la frecuencia, el RT[3] normalmente se ve reducido.

II.1.1 Valores típicos

Cada tipo de sala suele tener asociados un rango de valores característico. A continuación se citan algunos de ellos:

Campo libre[4] (exterior sin superficies reflectantes)	0.0 s
Habitación con muebles	0.3 – 0.5 s
Habitación vacía	0.6 – 0.8 s
Recinto para palabra	0.5 – 0.9 s
Ópera	1.3 – 1.7 s
Sala de conciertos	2.1 – 2.4 s
Catedral	> 2 s

Tabla 1: Valores típicos del tiempo de reverberación.

[2] En la *app* se trabaja con un parámetro muy similar al EDT, ya que es entre -5 y -15 dB.
[3] RT: Tiempo de Reverberación.
[4] Caso ideal.

III. METODOLOGÍA Y HERRAMIENTAS

El software ha sido implementado en Android con el IDE[5] Eclipse. Android se vale esencialmente de dos lenguajes de programación para el desarrollo de aplicaciones: Java y XML.

Adicionalmente, para la implementación de ciertos algoritmos, ha sido de gran utilidad MATLAB (en especial MATLAB Coder) y JNI[6]. Aunque por motivos de aprendizaje se haya preferido implementar finalmente todos los métodos íntegramente en Java.

III.1. Android

Android es un sistema operativo basado en Linux enfocado a smartphones y tablets.

Fig. 2: Logotipo de Android.

En el año 2005 Google compró Android Inc., la empresa que inicialmente había desarrollado el producto y empezaron a trabajar en la máquina Dalvik, una especie de máquina virtual similar a la de Java, pero optimizada para dispositivos móviles.

Dos años más tarde, surgió un consorcio formado por un conglomerado de empresas con el propósito de desarrollar estándares abiertos para móviles y se creó la primera versión del Android SDK[7].

En el año 2008 se lanzó el primer móvil con SO Android.

Desde entonces, el crecimiento de este sistema operativo ha sido exponencial.

La arquitectura de Android está formada por varias capas de software libre:

1. Aplicaciones.
2. Entorno de aplicación.
3. Máquina Dalvik.
4. Bibliotecas nativas.
5. Núcleo Linux.

Cada una de estas capas usa elementos de la de abajo para realizar sus funciones.

En el ámbito de la programación, se fundamenta en Java y XML, aunque también se puede desarrollar en C, C++ con al Android NDK[8].

[5] IDE: Integrated Development Environment. Entorno de desarrollo integrado.
[6] JNI: Java Native Interface. Interfaz Nativa de Java.
[7] SDK: Software Development Kit. Kit de Desarrollo de Software.
[8] NDK: Native Development Kit. Kit de Desarrollo Nativo.

III.1.1 Java

Java es un lenguaje de alto nivel multiplataforma. Es decir, puede ejecutarse en cualquier sistema operativo, siempre que se disponga de la máquina virtual que lo compila. Está basado principalmente en C, y utiliza programación orientada a objetos.

Es la herramienta principal de programación en Android.

III.1.2 XML

XML es un lenguaje estructurado mediante etiquetas que define un conjunto de reglas semánticas que facilitan la organización de información.

Se utiliza para la confección de pantallas y el manifiesto.

III.2. Eclipse

Eclipse es un IDE de código abierto multiplataforma.

Es el utilizado para la elaboración de este proyecto.

III.3. MATLAB

MATLAB es un entorno de programación de muy alto nivel enfocado sobre todo al cálculo numérico, con el que se consiguen resultados mucho más rápidamente que con otro tipo de lenguaje de nivel más bajo.

Es una herramienta imprescindible en ingeniería.

III.3.1 MATLAB Coder

MATLAB Coder es un plugin de MATLAB que permite extraer el código en C/C++ de gran cantidad de funciones.

Es de gran utilidad en la obtención de código nativo para realizar algoritmos con cierto grado de complejidad.

III.4. JNI

JNI es una interfaz que permite que un programa escrito en Java, pueda acceder a bibliotecas escritas en C/C++.

Muy útil en este proyecto para probar código antes de implementarlo.

III.4.1 C/C++

Son lenguajes de programación muy eficientes en los que están basados gran parte de los sistemas operativos. C es un lenguaje de medio nivel y C++ un súper conjunto de éste, que ya accede a características propias de los lenguajes orientados a objetos.

Se suelen utilizar para optimizar algoritmos debido a su gran rendimiento.

IV. ANÁLISIS DE REQUISITOS

Básicamente, la aplicación debe tener una pantalla con un botón como interfaz entre el usuario y el programa. Al pulsarlo una vez se pondrá en modo de grabación y al hacerlo de nuevo, mostrará los resultados.

También debe de poseer un modo convencional de salida de la aplicación desde el botón de menú del dispositivo.

V. DISEÑO DEL PROGRAMA

En este apartado del proyecto se detalla la ingeniería inversa del programa realizado.

V.1. Diseño de la clase RTActivity

Debido a la multitud de campos (atributos de la clase) implementados, únicamente, se muestran los métodos incluidos dentro de la clase.

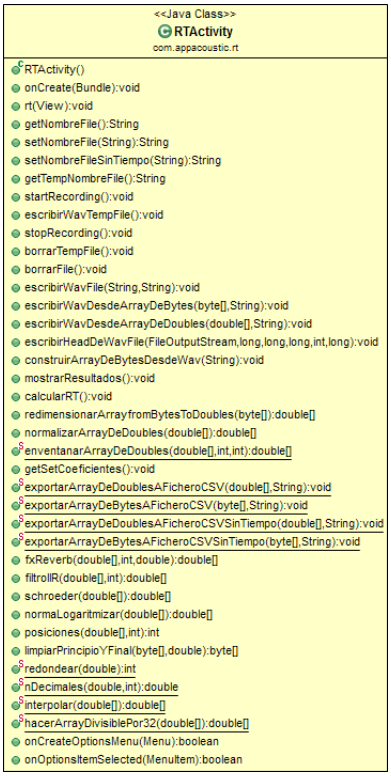

Fig. 3: Diseño UML[9] de la clase RTActivity (métodos).

[9] UML: Unified Modeling Language. Lenguaje de Modelado.

V.2. Diseño de las clases Butterworth

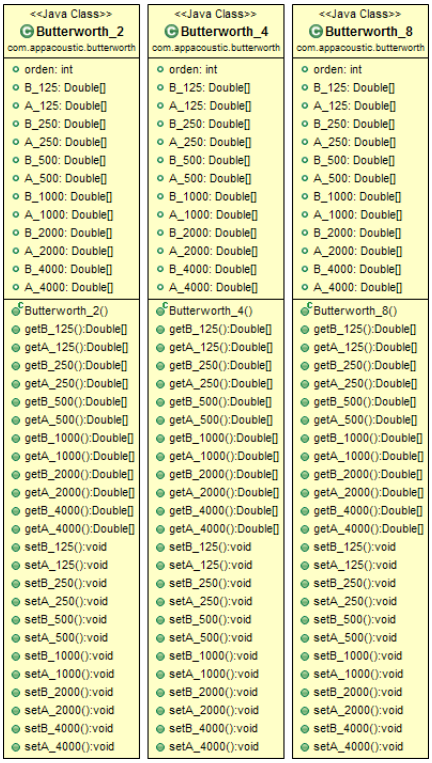

Fig. 4: Diseño UML de las clases Butterworth.

V.3. Relaciones entre clases

En la Fig. 5 se aprecia que la clase RTActivity hereda de la clase Activity y tiene objetos de las clases AudioRecord, TextView, Button, Thread, String y Double. Así como que usa las clases Butterworth.

A grandes rasgos, es una actividad con un botón y textos que usa un hilo para grabar.

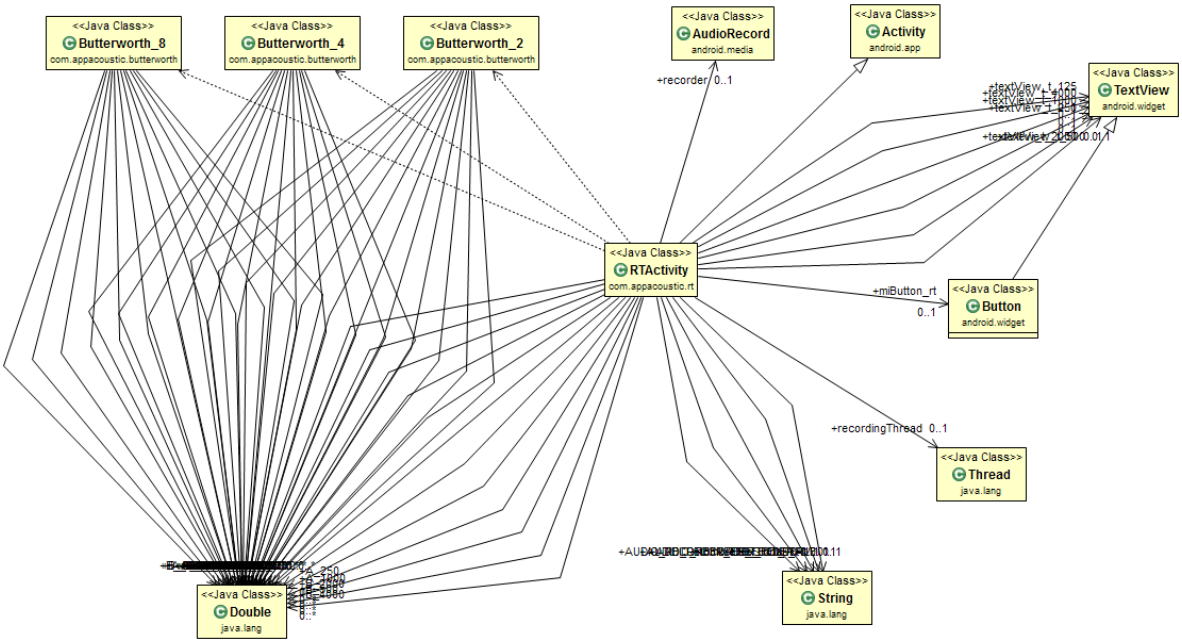

Fig. 5: Relaciones entre clases.

V.4. Algoritmo para calcular el tiempo de reverberación

Los pasos que sigue el software realizado en Java para medir y calcular el RT son los siguientes:

1. Grabar la señal.
2. Copiarla a un fichero *.wav temporal.
3. Construir un array de bytes a partir de este archivo.
4. Borrar el audio (liberación de memoria).
5. Llenar de ceros el principio y final del array para limpiarlo de datos indeseados.
6. Crear un array de doubles de la mitad de tamaño que el de bytes y rellenar cada casilla del de doubles con dos del de bytes.
7. Construir otro array de doubles con la mitad de tamaño del anterior, en el que almacenamos una de cada dos casillas de éste.
8. Enventanar el array actual en función del valor de las casillas (amplitud) obteniendo uno que sólo contenga la información significativa.
9. Eliminar casillas hasta que tenga un número de ellas múltiplo de 32, por motivos de velocidad de cálculo.
10. Normalizar el array entre -1 y 1.
11. Obtener los coeficientes de los filtros de Butterworth para las 6 bandas.
12. Filtrar obteniendo 6 arrays, cada uno representativo de su octava.
13. Hacer la integral de Schroeder a dichos arrays.
14. Normalizarlos.
15. Pasar los resultados a dB.
16. Calcular el tiempo de reverberación de cada uno de ellos.
17. Mostrar los resultados por pantalla.

VI. IMPLEMENTACIÓN

En este apartado se detalla la implementación realizada para la obtención de la *app*.

Al ser Android, se ha utilizado Java y XML, como hemos citado anteriormente.

La parte de Java, se ha ordenado en dos paquetes:

- `package com.appacoustic.butterworth;`
  - `Butterworth_2`
  - `Butterworth_4`
  - `Butterworth_8`
- `package com.appacoustic.rt;`
  - `RTActivity`

En el primer paquete se encuentran las clases donde se almacenan los coeficientes de los filtros y en la segunda, se ha incluido la actividad principal, que tiene los métodos que implementan las funciones de la app.

La parte de XML, son el AndroidManifest y el main.

VI.1. Butterworth_x

Las clases Butterworth (donde el x hace referencia al orden del filtro) son meros contenedores. Están compuestas por unos campos donde se almacenan los valores de los coeficientes conseguidos previamente con MATLAB y sus respectivos métodos getters y setters. (Ver anexo XI.1)

El script realizado para la obtención de dichos coeficientes es el siguiente:

```
clear, close all, clc
format longe
fs=44100;
orden=2;
frecuencias_limite = 1000*(2.^((-4:2)))*sqrt(2);
char=['125 Hz:';'250 Hz:';'500 Hz:';' 1 kHz:';' 2 kHz:';' 4 kHz:'];
for i=1:length(frecuencias_limite)-1
    disp(char(i,:))
    [B,A] = butter(orden,frecuencias_limite(i:i+1)/(fs/2))
    pause, clc
end
format
```

Donde se ha ido cambiando el orden (2, 4, 8) según fuera la clase en cuestión.

VI.2. RTActivity

Es la actividad principal. Desde esta clase se realizan todas las gestiones con el resto de componentes del programa.

Esta clase está basada en la siguiente función implementada en *MATLAB*:

```
function tiempo = reverbTotal(x)
%-------------------------------------------------------------------------
% "Calcula el tiempo de reverberación de una señal mono"
%
% tiempo = reverbTotal(x)
%
% x: Señal (array de doubles)
% tiempo: Tiempo de reverberación
%-------------------------------------------------------------------------
%% INICIALIZACIONES PARA EL CODER:
tiempo = 0;
y = zeros(176384, 6); % Lo inicilizamos al múltiplo de 32 más cercano a 176400 hacia abajo

%% fs, dBstart y dBend:
fs = 44100;
dBstart = -5; % ejemplo: -5 -15 -> EDT
dBend = -15;
aux = 0; % Inicializamos la variable auxiliar

%% REDIMENSIONAMOS LA SEÑAL "x" PARA QUE QUEDE con un número de muestras múltiplo de 32:
x = x(1:floor(length(x)/32)*32); % Simplemente se le quitan unas casillas del final

%% HACEMOS LA INTEGRAL DE SCHROEDER:
x = cumsum(x.^2); % Hacemos la suma acumulada (integral discreta) de la señal filtrada al cuadrado

%% NORMALIZAMOS Y PASAMOS A LOGARÍTMICO:
x = 10*log10(1-x/max(x));

%% POSICIONES FINAL E INICIAL (dB) PARA HACER EL CÁLCULO DEL RT:
[~,posi_dBend] = min(abs(x-dBend)); % ~ es para que vaya mejor
[~,posi_dBstart] = min(abs(x-dBstart));

tiempo = 60/(-dBend+dBstart)*(posi_dBend-posi_dBstart)/fs; % RT (Resultado final)

return % para que vaya el ejecutable (si es que se hace uno)
```

Esta función es una versión simplificada de lo que finalmente hace el programa, ya que no considera el cálculo de cada tiempo en función de la frecuencia, pero ha sido de mucha utilidad para extraer los algoritmos e implementarlos finalmente en Java.

La clase RTActivity está compuesta por multitud de campos y métodos. Los nombres de cada uno de ellos son descriptivos según su función. No obstante, en los anexos vienen adjuntos y explicados con detalle en el mismo programa. Se enumeran a continuación:

- Campos. (Ver anexo XI.22)
- Métodos:

onCreate()	(Ver anexo XI.3.1)
rt()	(Ver anexo XI.3.2)
getNombreFile()	(Ver anexo XI.3.3)
setNombreFile()	(Ver anexo XI.3.4)
setNombreFileSinTiempo()	(Ver anexo XI.3.5)
getTempNombreFile()	(Ver anexo XI.3.6)
startRecording()	(Ver anexo XI.3.7)
escribirWavTempFile()	(Ver anexo XI.3.8)
stopRecording()	(Ver anexo XI.3.9)
borrarTempFile()	(Ver anexo XI.3.10)
borrarFile()	(Ver anexo XI.3.11)
escribirWavFile()	(Ver anexo XI.3.12)
escribirWavDesdeArrayDeBytes()	(Ver anexo XI.3.13)
escribirWavDesdeArrayDeDoubles()	(Ver anexo XI.3.14)
escribirHeadDeWavFile()	(Ver anexo XI.3.15)
construirArrayDeBytesDesdeWav()	(Ver anexo XI.3.16)
mostrarResultados()	(Ver anexo XI.3.17)
calcularRT()	(Ver anexo XI.3.18)
redimensionarArrayfromBytesToDoubles()	(Ver anexo XI.3.19)
normalizarArrayDeDoubles()	(Ver anexo XI.3.20)
enventanarArrayDeDoubles()	(Ver anexo XI.3.21)
getSetCoeficientes()	(Ver anexo XI.3.22)
exportarArrayDeDoublesAFicheroCSV()	(Ver anexo XI.3.23)
exportarArrayDeBytesAFicheroCSV()	(Ver anexo XI.3.24)
exportarArrayDeDoublesAFicheroCSVSinTiempo()	(Ver anexo XI.3.25)
exportarArrayDeBytesAFicheroCSVSinTiempo()	(Ver anexo XI.3.26)
fxReverb()	(Ver anexo XI.3.27)
filtroIIR()	(Ver anexo XI.3.28)
schroeder()	(Ver anexo XI.3.29)
normaLogaritmizar()	(Ver anexo XI.3.30)
posiciones()	(Ver anexo XI.3.31)
limpiarPrincipioYFinal()	(Ver anexo XI.3.32)
redondear()	(Ver anexo XI.3.33)
nDecimales()	(Ver anexo XI.3.34)
interpolar()	(Ver anexo XI.3.35)
hacerArrayDivisiblePor32()	(Ver anexo XI.3.36)
onCreateOptionsMenu()	(Ver anexo XI.3.37)
onOptionsItemSelected()	(Ver anexo XI.3.38)

Tabla 2: Métodos implementados en RTActivity.

VI.3. AndroidManifest

Es la parte donde indicamos la versión del programa, permisos que se necesitan, actividades existentes, etc.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.appacoustic.rt"
    android:versionCode="1"
    android:versionName="1.0.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".RTActivity"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.NoTitleBar"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

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

</manifest>
```

En este caso se dan las siguientes características:

1. Es la primera versión.
2. La versión mínima de Android es la 2.2 (Froyo).
3. Existe una actividad (RTActivity).
4. No hay barra de título.
5. La pantalla no gira y está en vertical.
6. Se necesitan permisos de escritura en el almacenamiento externo (normalmente SD) y grabación de audio.

VI.4. main

Se trata de la pantalla principal del software. En esta primera versión, es la única existente, y desde ella se realiza toda la interactuación con el usuario.

VII. MANUAL DEL USUARIO

AppAcoustiC RT 1.0

Aplicación para calcular el tiempo de reverberación en bandas de octava.

VII.1. Instalación

Copiar el archivo RT.apk a la memoria SD del terminal e instalar.

VII.2. Funcionamiento

VII.2.1 Arrancar la aplicación.

Fig. 6: Botón de inicio de la aplicación.

VII.2.2 Apretar START.

Fig. 7: Pantalla de inicio.

VII.2.3 Dar una palmada.

Fig. 8: Palmada.

VII.2.4 Apretar STOP.

Fig. 9: Pantalla para acabar de realizar la medición.

VII.2.5 Darle a "Sí" si la medida ha sido correcta.

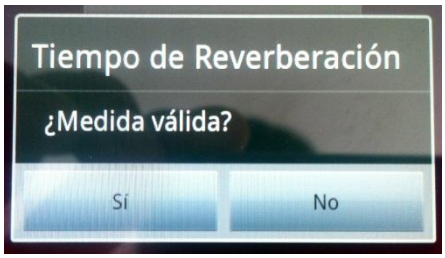

Fig. 10: Validación de la medida.

Para conseguir unos resultados óptimos, es más que recomendable tratar de situar el dispositivo lejos de superficies reflectantes y pegar la palmada a mayor distancia del radio crítico, para que el sonido reverberado sea mayor que el directo.

Si el tiempo de medida ha sido demasiado corto, aparecerá un mensaje en la pantalla indicándolo.

La app está disponible en español e inglés. Se dispondrá automáticamente en un idioma u otro según la configuración que tengamos en el sistema operativo.

Para salir de la aplicación pulsar la tecla de menú y apretar Salir.

Fig. 11: Pantalla de salida.

VIII. CONCLUSIONES Y LÍNEAS FUTURAS

El hecho de realizar una aplicación para este sistema operativo, era con el afán de conocimiento y poder aplicarlo a un entorno laboral. Es un campo en el que actualmente hay una demanda de profesionales creciente ya que se necesitan programadores que dominen los lenguajes y las herramientas para llevar a cabo este tipo de aplicaciones.

Dependiendo de la calidad del micrófono y/o infraestructura de audio en recepción del dispositivo, se tendrán unos resultados más o menos fidedignos. En el caso del modelo *Sony Ericsson Xperia Neo*, con el cual se han realizado las primeras pruebas, únicamente se obtienen resultados fiables a partir de 1 kHz.

Sería interesante probar micrófonos de precisión periféricos.

En breve, esta aplicación estará disponible en el Google Play.

Como línea futura, se destaca que éste es el primer proyecto de una familia de aplicaciones para móviles en el ámbito acústico.

IX. AGRADECIMIENTOS

A mi buen amigo, gran persona y excelentísimo profesor, Javier Redondo.

A Jordi Bataller, por sus inestimables consejos sobre programación.

A Miguel Ferrer, por la confianza depositada hacia mi persona.

A mis grandes compañeros de piso, Juanri y Andrés.

A mis queridos compañeros del Máster.

A todo el equipo M&M Publicidad.

Y especialmente, a tod@s y cada un@ de los buen@s amig@s que he hecho a lo largo de todos estos años en Gandía.

X. REFERENCIAS

[1] Ableson, F., Collins, C., & Sen, R. (2011). *Android: guía para desarrolladores.* Madrid: Anaya Multimedia.

[2] AENOR. (Abril de 2001). Medición del tiempo de reverberación de recintos con referencia a otros parámetros acústicos. Madrid.

[3] AENOR. (Diciembre de 2008). Medición de parámetros acústicos en recintos. *Tiempo de reverberación en recintos ordinarios*. Madrid.

[4] AENOR. (Septiembre de 2009). Medición de parámetros acústicos en recintos. *Tiempo de reverberación en recintos ordinarios*. Madrid.

[5] AENOR. (Febrero de 2010). Medición de parámetros acústicos en recintos. *Salas de espectáculos*. Madrid.

[6] Brinkmann, P. (2012). *Making musical apps.* Sebastopol: O'reilly.

[7] Garrido, A. (1997). *Principios de acústica.* Madrid: Sanz y Torres.

[8] Hughes, J. (2011). *Marketing de aplicaciones Android.* Madrid: Anaya Multimedia.

[9] Kinsler, L. E. (1995). *Fundamentos de Acústica.* México: Limusa.

[10] Lee, W. (2012). *Android 4. Desarrollo de aplicaciones.* Madrid: Anaya Multimedia.

[11] Llinares Galiana, J., Llopis Reina, A., & Sancho, J. (1996). *Acústica arquitectónica y urbanística.* Valencia: SPUPV.

[12] Mednicks, Z., Dornin, L., Blake Meike, G., & Nakamura, M. (2011). *Programming Android.* Beijing: O'Reilly.

[13] Pueo Ortega, B., & Romá Romero, M. (2003). *Electroacústica: Altavoces y micrófonos.* Madrid: Pearson Prentice Hall.

[14] Recuero López, M. (1994). *Ingeniería Acústica.* Madrid: Paraninfo.

[15] Recuero López, M., & Gil, C. (1992). *Acústica Arquitectónica.* Madrid: Ártica.

[16] *Stackoverflow*. (s.f.). Recuperado el 2012, de http://stackoverflow.com/

[17] Stark, J. (2010). *Building android apps with HTML, CSS, and JavaScript.* Farnham: O'Reilly.

[18] Tomás Gironés, J. (2011). *El gran libro de Android.* Barcelona: Marcombo.

[19] Zechner, M. (2011). *Desarrollo de juegos para Android.* Madrid: Anaya Multimedia.

XI. ANEXOS

XI.1. Butterworth_2

Sólo se ha incluido la clase Butterworth_2, ya que las otras dos son muy similares.

```
package com.appacoustic.butterworth;

public class Butterworth_2 {
        /*
         * N=2. Vectores de coeficientes de 2*2+1 = 5 elementos
         *
         * Se usan tantos decimales en los coeficientes de Buttherworth porque podríamos enfrentarnos con
NaN (Not a Number)
         * si pusieramos pocos, ya que la falta de resolución puede ocasionar problemas tipo división por
cero
         *
         * En versiones posteriores del paquete hay que:
         *      - Aprovechar la simetría de los coeficientes del numerador (B) para ahorrar memoria
         *      - Implementar la obtención de coeficientes
         */

        public int orden = 2;

        public Double B_125[];
    public Double A_125[];
    public Double B_250[];
    public Double A_250[];
    public Double B_500[];
    public Double A_500[];
    public Double B_1000[];
    public Double A_1000[];
    public Double B_2000[];
    public Double A_2000[];
    public Double B_4000[];
    public Double A_4000[];

    // getters
        public Double[] getB_125() {
                return B_125;
        }
        public Double[] getA_125() {
                return A_125;
        }
        public Double[] getB_250() {
                return B_250;
        }
        public Double[] getA_250() {
                return A_250;
        }
        public Double[] getB_500() {
                return B_500;
        }
        public Double[] getA_500() {
                return A_500;
        }
        public Double[] getB_1000() {
                return B_1000;
        }
        public Double[] getA_1000() {
                return A_1000;
        }
        public Double[] getB_2000() {
                return B_2000;
        }
        public Double[] getA_2000() {
                return A_2000;
        }
        public Double[] getB_4000() {
                return B_4000;
        }
        public Double[] getA_4000() {
```

```
        return A_4000;
}

// setters
public void setB_125() {
        B_125 = new Double[orden*2+1];

        B_125[0]=3.929676515700289e-05;
        B_125[1]=(double) 0;
        B_125[2]=-7.859353031400579e-05;
        B_125[3]=(double) 0;
        B_125[4]=3.929676515700289e-05;
}

public void setA_125() {
        A_125 = new Double[orden*2+1];

        A_125[0]=(double) 1;
        A_125[1]=-3.981559459679419e+00;
        A_125[2]=5.945472787148730e+00;
        A_125[3]=-3.946261397152608e+00;
        A_125[4]=9.823481693904715e-01;
}

public void setB_250() {
        B_250 = new Double[orden*2+1];

        B_250[0]=1.558059149882655e-04;
        B_250[1]=(double) 0;
        B_250[2]=-3.116118299765310e-04;
        B_250[3]=(double) 0;
        B_250[4]=1.558059149882655e-04;
}

public void setA_250() {
        A_250 = new Double[orden*2+1];

        A_250[0]=(double) 1;
        A_250[1]=-3.961870007021670e+00;
        A_250[2]=5.888793913787786e+00;
        A_250[3]=-3.891930251948569e+00;
        A_250[4]=9.650079262290459e-01;
}

public void setB_500() {
        B_500 = new Double[orden*2+1];

        B_500[0]=6.124167588371880e-04;
        B_500[1]=(double) 0;
        B_500[2]=-1.224833517674376e-03;
        B_500[3]=(double) 0;
        B_500[4]=6.124167588371880e-04;
}

public void setA_500() {
        A_500 = new Double[orden*2+1];

        A_500[0]=(double) 1;
        A_500[1]=-3.918823751633717e+00;
        A_500[2]=5.769261524124245e+00;
        A_500[3]=-3.781653237866402e+00;
        A_500[4]=9.312403077020502e-01;
}

public void setB_1000() {
        B_1000 = new Double[orden*2+1];

        B_1000[0]=2.366944651196807e-03;
        B_1000[1]=(double) 0;
        B_1000[2]=-4.733889302393613e-03;
        B_1000[3]=(double) 0;
        B_1000[4]=2.366944651196807e-03;
}
```

```
	public void setA_1000() {
		A_1000 = new Double[orden*2+1];

		A_1000[0]=(double) 1;
		A_1000[1]=-3.818619281298950e+00;
		A_1000[2]=5.507523751882116e+00;
		A_1000[3]=-3.555730199154567e+00;
		A_1000[4]=8.672088097654537e-01;
	}

	public void setB_2000() {
		B_2000 = new Double[orden*2+1];

		B_2000[0]=8.861384842871868e-03;
		B_2000[1]=(double) 0;
		B_2000[2]=-1.772276968574374e-02;
		B_2000[3]=(double) 0;
		B_2000[4]=8.861384842871868e-03;
	}

	public void setA_2000() {
		A_2000 = new Double[orden*2+1];

		A_2000[0]=(double) 1;
		A_2000[1]=-3.566233905008885e+00;
		A_2000[2]=4.910270117609992e+00;
		A_2000[3]=-3.090416995530277e+00;
		A_2000[4]=7.520594824636435e-01;
	}

	public void setB_4000() {
		B_4000 = new Double[orden*2+1];

		B_4000[0]=3.135695184246452e-02;
		B_4000[1]=(double) 0;
		B_4000[2]=-6.271390368492903e-02;
		B_4000[3]=(double) 0;
		B_4000[4]=3.135695184246452e-02;
	}

	public void setA_4000() {
		A_4000 = new Double[orden*2+1];

		A_4000[0]=(double) 1;
		A_4000[1]=-2.889193822757614e+00;
		A_4000[2]=3.560473220464110e+00;
		A_4000[3]=-2.159920874581686e+00;
		A_4000[4]=5.658008519099556e-01;
	}

} // () Butterworth_2
```

(Volver al apartado VI.1)

XI.2. Campos RTActivity

```
public static final int RECORDER_BPP = 16; // 16 bits por muestra
    public static final String AUDIO_RECORDER_FILE_EXT_WAV = ".wav"; // Extensión del archivo de audio a
grabar
    public static final String AUDIO_RECORDER_FOLDER = "RT"; // Directorio de la tarjeta SD del dispositivo
móvil
    public static final String CSV_FOLDER = "RT/CSV"; // Directorio donde exportaremos los arrays
    public static final String AUDIO_RECORDER_TEMP_FILE = "record_temp.raw"; // Nombre del archivo temporal
    public static final int RECORDER_SAMPLERATE = 44100; // Frecuencia de muestreo
    public static final int RECORDER_CHANNELS = AudioFormat.CHANNEL_IN_MONO; // 1 canal (micrófono simple)
    public static final int RECORDER_AUDIO_ENCODING = AudioFormat.ENCODING_PCM_16BIT; // Codificación PCM
de 16 bits por muestra

    public AudioRecord recorder; // Declaramos el objeto necesario para hacer grabaciones
    public int bufferSize; // Tamaño del buffer
    public Thread recordingThread; // Hilo para hacer la grabación
    public boolean isRecording = false; // Booleano que dice si se está grabando o no (empieza en false,
lógicamente)
    public boolean haPulsado = true; // Tiene que empezar en "true", porque una vez se ha metido dentro de
rt() tiene que entrar dentro del "if"
    public Button miButton_rt; // Declaramos el botón principal
    public long totalAudioLen; // Necesito que sea global para escribir el head en el wav que escribo a
partir del array de bytes
    public byte[] x_byte; // Este es el array en el que cargamos todo el archivo de audio no le daremos
ningún tamaño, ya que dependerá de la duración de la grabación (stream)
    public String nombreFileGlobal; // Nombre del fichero que copiamos a partir del temporal
    public int duracion; // Contador para que no acabe de grabar antes de tiempo

    // Valores para calcular el tiempo de reverberación a partir del número de muestras que hay entre las
posiciones que poseen estas amplitudes. Ej: EDT (-5, -15 [dB])
    public int dBStart = -5; // Amplitudes (En versiones posteriores se hará una interfaz para que el
usuario pueda cambiar estos valores)
    public int dBEnd = -15;

    public int posicion_dBStart_125; // Posiciones
    public int posicion_dBEnd_125;
    public int posicion_dBStart_250;
    public int posicion_dBEnd_250;
    public int posicion_dBStart_500;
    public int posicion_dBEnd_500;
    public int posicion_dBStart_1000;
    public int posicion_dBEnd_1000;
    public int posicion_dBStart_2000;
    public int posicion_dBEnd_2000;
    public int posicion_dBStart_4000;
    public int posicion_dBEnd_4000;

        // Coeficientes del numerador y denominador de todas las bandas (Son Double con mayúscula (Clase
Double) para poder trabajar mejor)
    public Double[] B_125;
    public Double[] A_125;
    public Double[] B_250;
    public Double[] A_250;
    public Double[] B_500;
    public Double[] A_500;
    public Double[] B_1000;
    public Double[] A_1000;
    public Double[] B_2000;
    public Double[] A_2000;
    public Double[] B_4000;
    public Double[] A_4000;

    // Según el orden del filtro que estemos usando tendremos un recorrido mayor o menor para evitar la
inestabilidad inherente en los filtros IIR.
    // Orden 8: 2*8+1=17 --- Orden 4: 2*4+1=9
    public int tam_125;
    public int tam_250;
    public int tam_500;
    public int tam_1000;
    public int tam_2000;
    public int tam_4000;
```

```
    // Inicializamos los tiempos de reverberación
    public double reverbTime_125 = 0.00;
    public double reverbTime_250 = 0.00;
    public double reverbTime_500 = 0.00;
    public double reverbTime_1000 = 0.00;
    public double reverbTime_2000 = 0.00;
    public double reverbTime_4000 = 0.00;

    // Declaramos los TextView donde irán los tiempos de reverberación
    public TextView textView_t_125;
    public TextView textView_t_250;
    public TextView textView_t_500;
    public TextView textView_t_1000;
    public TextView textView_t_2000;
    public TextView textView_t_4000;
```

(Volver al apartado VI.2)

XI.3. Métodos RTActivity

XI.3.1 onCreate()

```
@Override
    public void onCreate(Bundle savedInstanceState) { // Lo que se carga nada más arrancar al app
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        bufferSize = AudioRecord.getMinBufferSize(RECORDER_SAMPLERATE, RECORDER_CHANNELS,
RECORDER_AUDIO_ENCODING); // Calculamos el tamaño mínimo que tiene que tener el buffer (4096 bytes)
        miButton_rt = (Button) findViewById(R.id.button_rt); // Relaciono con el botón para poder cambiarle
el texto después

        // Relacionamos con los TextView de los tiempos de reverberación
        textView_t_125 = (TextView) findViewById(R.id.textView_t_125);
        textView_t_250 = (TextView) findViewById(R.id.textView_t_250);
        textView_t_500 = (TextView) findViewById(R.id.textView_t_500);
        textView_t_1000 = (TextView) findViewById(R.id.textView_t_1000);
        textView_t_2000 = (TextView) findViewById(R.id.textView_t_2000);
        textView_t_4000 = (TextView) findViewById(R.id.textView_t_4000);

    } // onCreate()
```

(Volver a la Tabla 2)

XI.3.2 rt()

```
public void rt(View v){ // Acciones del botón principal
        // Le damos como parámetro de entrada un View porque se ejecuta al llamar a onClick(), que hereda
de la clase View
        if (haPulsado){
                miButton_rt.setText("STOP"); // Le cambio el texto a STOP
                haPulsado = false;
                duracion=0; // Inicializamos el contador de la grabación
                startRecording();
        }else{
                miButton_rt.setText("START"); // Le cambio el texto a START
                haPulsado = true;
                if(duracion>10){
                        stopRecording();
                mostrarResultados(); // Saca un cuadro de diálogo que nos pregunta si la medida es válida,
para procesarla o no
                //Toast.makeText(getApplicationContext(), "Duracion de la grabación: "+duracion,
Toast.LENGTH_SHORT).show();
                }else{
                        Toast.makeText(getApplicationContext(),R.string.repita, Toast.LENGTH_SHORT).show();
// Hay que repetir la medida
                        stopRecording();
                }

        }
    } // rt()
```

(Volver a la Tabla 2)

XI.3.3 getNombreFile()

```
public String getNombreFile(){ // Devuelve el nombre del fichero de audio
        String filepath = Environment.getExternalStorageDirectory().getPath(); // Obtenemos la ruta donde
se alojará el fichero de audio
        File file = new File(filepath, AUDIO_RECORDER_FOLDER);

        if(!file.exists()){
                file.mkdirs(); // Si no existe el directorio, lo creamos
        }

        nombreFileGlobal = (file.getAbsolutePath() + "/" + System.currentTimeMillis() +
AUDIO_RECORDER_FILE_EXT_WAV); // Devuelve el nombre del archivo: Ruta completa + fecha y hora (en
milisegundos desde 1970) + extensión

        return nombreFileGlobal;
    } // getNombreFile()
```

(Volver a la Tabla 2)

XI.3.4 setNombreFile()

```
public String setNombreFile(String nombre){ // Setea el nombre de un archivo de audio (añade milisegundos
desde 1970)
        String filepath = Environment.getExternalStorageDirectory().getPath(); // Obtenemos la ruta donde
se alojará el fichero de audio
        File file = new File(filepath, AUDIO_RECORDER_FOLDER);

        if(!file.exists()){
                file.mkdirs(); // Si no existe el directorio, lo creamos
        }

        return (file.getAbsolutePath() + "/" + System.currentTimeMillis() + nombre +
AUDIO_RECORDER_FILE_EXT_WAV); // Devuelve el nombre del archivo: Ruta completa + fecha y hora (en
milisegundos desde 1970) + nombre + extensión
    } // setNombreFile()
```

(Volver a la Tabla 2)

XI.3.5 setNombreFileSinTiempo()

```
public String setNombreFileSinTiempo(String nombre){ // Setea el nombre de un archivo de audio
        String filepath = Environment.getExternalStorageDirectory().getPath(); // Obtenemos la ruta donde
se alojará el fichero de audio
        File file = new File(filepath, AUDIO_RECORDER_FOLDER);

        if(!file.exists()){
                file.mkdirs(); // Si no existe el directorio, lo creamos
        }

        return (file.getAbsolutePath() + "/" + nombre + AUDIO_RECORDER_FILE_EXT_WAV); // Devuelve el nombre
del archivo: Ruta completa + nombre + extensión
    } // setNombreFileSinTiempo()
```

(Volver a la Tabla 2)

XI.3.6 getTempNombreFile()

```
public String getTempNombreFile(){ // Devuelve el nombre del fichero de audio temporal
        String filepath = Environment.getExternalStorageDirectory().getPath(); // Obtenemos la ruta donde
se alojará el fichero de audio
        File file = new File(filepath, AUDIO_RECORDER_FOLDER);

        if(!file.exists()){
                file.mkdirs(); // Si no existe el directorio, lo creamos
        }

        File tempFile = new File(filepath, AUDIO_RECORDER_TEMP_FILE); // Directorio donde se alojará el
fichero de audio temporal

        if(tempFile.exists())
                tempFile.delete(); // Machaca el directorio anterior si existe (es temporal)

        return (file.getAbsolutePath() + "/" + AUDIO_RECORDER_TEMP_FILE); // Devuelve el nombre del archivo
temporal
    } // getTempNombreFile()
```

(Volver a la Tabla 2)

XI.3.7 startRecording()

```
public void startRecording(){ // Empieza a grabar
        // Creamos un objeto de la clase AudioRecord para poder usar hacer una grabación
        recorder = new AudioRecord(MediaRecorder.AudioSource.MIC, // El micrófono es la fuente sonora
                                                RECORDER_SAMPLERATE, // Frecuencia de muestreo del
grabador
                                                RECORDER_CHANNELS, // Nº de canales
                                                RECORDER_AUDIO_ENCODING, // Codificación PCM de 16 bits
por muestra
                                                bufferSize); // Tamaño del buffer
        recorder.startRecording(); // Empezamos a grabar
        Log.d("startRecording()", "Grabando...");
        isRecording = true; // Bandera de que SÍ estamos grabando
        recordingThread = new Thread(new Runnable() { // Hilo para hacer la grabación
                @Override
                public void run() {
                        escribirWavTempFile(); // Escribimos el audio en un fichero
                }
        },"AudioRecorder Thread");
        recordingThread.start(); // Comenzamos el hilo
        Log.d("startRecording()", "Escribiendo el fichero de audio...");
    } // startRecording()
```

(Volver a la Tabla 2)

XI.3.8 escribirWavTempFile()

```
public void escribirWavTempFile(){ // Método para escribir el audio en un fichero
        String fileNombre = getTempNombreFile(); // Pillamos el archivo temporal
        FileOutputStream os = null; // Inializamos el flujo de salida de datos

        try {
                os = new FileOutputStream(fileNombre); // Nos preparamos para escribir en el fichero
        } catch (FileNotFoundException e) {
                e.printStackTrace();
        }

        int read = 0; // Inicializamos read (es como si fuera un booleano)
        byte[] data = new byte[bufferSize]; // Buffer local que usamos para guardar el audio

        if(null != os){
                while(isRecording){ // isRecording es TRUE hasta que paramos de grabar
                        read = recorder.read(data, 0, bufferSize);
                        duracion++; // Vamos incrementando la duración
                        if(AudioRecord.ERROR_INVALID_OPERATION != read){
                                try {
                                        os.write(data); // Escribimos ayudándonos del buffer
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }
                        }
                }

                try {
                        os.close(); // Cerramos el flujo del wav
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
    } // escribirWavTempFile()
```

(Volver a la Tabla 2)

XI.3.9 stopRecording()

```
public void stopRecording(){ // Método para parar de grabar
        if(null != recorder){
                isRecording = false; // Bandera de que NO estamos grabando

                recorder.stop(); // Paramos de grabar
                Log.d("stopRecording()", "Hemos parado de grabar");

                recorder.release(); // Dejamos de usar el objeto (lo liberamos)
                recorder = null; // Reseteamos el objeto
                recordingThread = null; // Reseteamos el hilo
        }

        escribirWavFile(getTempNombreFile(),getNombreFile()); // Copiamos el fichero temporal
Log.d("stopRecording()", "Hemos copiado el fichero desde el archivo temporal al archivo que usamos");
        construirArrayDeBytesDesdeWav(getTempNombreFile()); // Ahora copiamos el el archivo temporal a un
array
        borrarTempFile(); // Borramos el fichero temporal ya que ya lo hemos gastado
    } // stopRecording()
```

(Volver a la Tabla 2)

XI.3.10 borrarTempFile()

```
public void borrarTempFile(){ // Método para borrar el fichero temporal
        File file = new File(getTempNombreFile());

        file.delete();
    } // borrarTempFile()
```

(Volver a la Tabla 2)

XI.3.11 borrarFile()

```
public void borrarFile(){ // Método para borrar el fichero de audio guardado en la SD
        File file = new File(nombreFileGlobal);

        file.delete();
    } // borrarFile()
```

(Volver a la Tabla 2)

XI.3.12 escribirWavFile()

```
public void escribirWavFile(String inFileNombre,String outFileNombre){ // Método para copiar el fichero de
audio
        // Le damos la entrada (archivo temporal) y la salida (archivo en el directorio bueno)
        FileInputStream in = null; // Inicializamos el flujo de datos de entrada
        FileOutputStream out = null; // Inicializamos el flujo de datos de salida
        long longSampleRate = RECORDER_SAMPLERATE; // frecuencia de muestreo
        int channels = 1; // El número de canales es 1 al ser una señal mono
        long byteRate = RECORDER_BPP * RECORDER_SAMPLERATE * channels/8; // Tasa de bits

        byte[] data = new byte[bufferSize]; // Buffer para poder copiar el audio en formato byte

        try {
                in = new FileInputStream(inFileNombre); // entrada
                out = new FileOutputStream(outFileNombre); // salida
                totalAudioLen = in.getChannel().size(); // Longitud del archivo de audio (duración).
Cargamos el tamaño del archivo para poder usarlo después en construirArrayDeBytesDesdeWav()
                long totalDataLen = totalAudioLen + 36; // El archivo de datos tiene 36 "casillas" más

                //Toast.makeText(getApplicationContext(), "Tamaño del archivo capturado: " +
totalDataLen/1024 + " KB", Toast.LENGTH_SHORT).show(); // Mostramos el tamaño

                // Escribimos el encabezamiento del fichero
                escribirHeadDeWavFile(out,
                                                totalAudioLen,
                                                totalDataLen,
                                                longSampleRate,
                                                channels,
                                                byteRate);

                while(in.read(data) != -1){
                        out.write(data);
                in.close();
                out.close();
        } catch (FileNotFoundException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
    } // escribirWavFile()
```

(Volver a la Tabla 2)

XI.3.13 escribirWavDesdeArrayDeBytes()

```
public void escribirWavDesdeArrayDeBytes(byte[] array, String outFileNombre){ // Método para copiar el
audio desde el array de bytes
        FileOutputStream out = null; // Inicializamos el flujo de datos de salida (wav)

        long longSampleRate = RECORDER_SAMPLERATE; // frecuencia de muestreo
        int channels = 1; // El número de canales es 1 al ser una señal mono
        long byteRate = RECORDER_BPP * RECORDER_SAMPLERATE * channels/8; // Tasa de bits

        try {
             out = new FileOutputStream(outFileNombre); // salida

                long totalDataLen = totalAudioLen + 36; // Lo saco de antes, ya que es global

                //Toast.makeText(getApplicationContext(), "Tamaño del archivo escrito a partir del array de
bytes: " + totalDataLen/1024 + " KB", Toast.LENGTH_SHORT).show(); // Mostramos el tamaño

                // Escribimos el encabezamiento del fichero
                escribirHeadDeWavFile(out,
                                                totalAudioLen,
                                                totalDataLen,
                                                longSampleRate,
                                                channels,
                                                byteRate);

                out.write(array); // Escribimos el fichero de audio con el array de bytes
                out.close(); // cerramos el flujo

        } catch (FileNotFoundException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
    } // escribirWavDesdeArrayDeBytes()
```

(Volver a la Tabla 2)

XI.3.14 escribirWavDesdeArrayDeDoubles()

```
public void escribirWavDesdeArrayDeDoubles(double[] array, String outFileNombre){ // Método para copiar el audio desde el array de doubles

        FileOutputStream out = null; // Inicializamos el flujo de datos de salida (wav)

        long totalAudioLen = 0; // Longitud del archivo de audio (duración)
        long totalDataLen = totalAudioLen + 36; // El archivo de datos tiene 36 "casillas" más
        long longSampleRate = RECORDER_SAMPLERATE; // frecuencia de muestreo
        int channels = 1; // El número de canales es 1 al ser una señal mono
        long byteRate = RECORDER_BPP * RECORDER_SAMPLERATE * channels/8; // Tasa de bits

        try {
             out = new FileOutputStream(outFileNombre); // salida

                totalAudioLen = array.length; // Lo calculamos de un modo alternativo (sigue siendo el tamaño a fin de cuentas)
                totalDataLen = array.length+36;

                //Toast.makeText(getApplicationContext(), "Tamaño del archivo escrito a partir del array de doubles: " + totalDataLen/1024 + " KB", Toast.LENGTH_SHORT).show(); // Mostramos el tamaño

                // Escribimos el encabezamiento del fichero
                escribirHeadDeWavFile(out,
                                                totalAudioLen,
                                                totalDataLen,
                                                longSampleRate,
                                                channels,
                                                byteRate);

                // Convertimos a bytes, ya que el flujo de salida no admite doubles
                int n;
                byte array_convertido_en_bytes[] = new byte[array.length];
                for (n=0;n<array.length;n=n+1){
                        array_convertido_en_bytes[n] = (byte) array[n];
                }

                out.write(array_convertido_en_bytes); // Escribimos el fichero de audio con el array de doubles(sin buffer ni nada)
                out.close(); // cerramos el flujo

        } catch (FileNotFoundException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
    } // escribirWavDesdeArrayDeDoubles()
```

(Volver a la Tabla 2)

XI.3.15 escribirHeadDeWavFile()

```
public void escribirHeadDeWavFile(FileOutputStream out, long totalAudioLen, long totalDataLen, long
longSampleRate, int channels, long byteRate) throws IOException{ // Método para escribir el encabezamiento
del fichero

    byte[] header = new byte[44];

    header[0] = 'R';  // Encabezamiento RIFF/WAVE
    header[1] = 'I';
    header[2] = 'F';
    header[3] = 'F';
    header[4] = (byte) (totalDataLen & 0xff);
    header[5] = (byte) ((totalDataLen >> 8) & 0xff);
    header[6] = (byte) ((totalDataLen >> 16) & 0xff);
    header[7] = (byte) ((totalDataLen >> 24) & 0xff);
    header[8] = 'W';
    header[9] = 'A';
    header[10] = 'V';
    header[11] = 'E';
    header[12] = 'f';  // 'fmt ' Trozo
    header[13] = 'm';
    header[14] = 't';
    header[15] = ' ';
    header[16] = 16;  // 4 bytes: tamaño del trozo 'fmt '
    header[17] = 0;
    header[18] = 0;
    header[19] = 0;
    header[20] = 1;  // formato = 1
    header[21] = 0;
    header[22] = (byte) channels;
    header[23] = 0;
    header[24] = (byte) (longSampleRate & 0xff);
    header[25] = (byte) ((longSampleRate >> 8) & 0xff);
    header[26] = (byte) ((longSampleRate >> 16) & 0xff);
    header[27] = (byte) ((longSampleRate >> 24) & 0xff);
    header[28] = (byte) (byteRate & 0xff);
    header[29] = (byte) ((byteRate >> 8) & 0xff);
    header[30] = (byte) ((byteRate >> 16) & 0xff);
    header[31] = (byte) ((byteRate >> 24) & 0xff);
    header[32] = (byte) (2 * 16 / 8);  // Bloqueamos la alineación
    header[33] = 0;
    header[34] = RECORDER_BPP;  // bits por muestra
    header[35] = 0;
    header[36] = 'd';
    header[37] = 'a';
    header[38] = 't';
    header[39] = 'a';
    header[40] = (byte) (totalAudioLen & 0xff);
    header[41] = (byte) ((totalAudioLen >> 8) & 0xff);
    header[42] = (byte) ((totalAudioLen >> 16) & 0xff);
    header[43] = (byte) ((totalAudioLen >> 24) & 0xff);

    out.write(header, 0, 44);
    } // escribirHeadDeWavFile
```

(Volver a la Tabla 2)

XI.3.16 construirArrayDeBytesDesdeWav()

```
public void construirArrayDeBytesDesdeWav(String inFileNombre){ // Método para copiar el array desde el
audio
        FileInputStream in = null; // Flujo de datos de entrada (el archivo de audio)
        ByteArrayOutputStream out = null; // Flujo de datos de salida (para guardar el array)

        byte[] data = new byte[bufferSize]; // Buffer para el array

        try{
                in = new FileInputStream(inFileNombre); // Instanciamos el flujo de salida del array
                out = new ByteArrayOutputStream(); // Instanciamos el flujo de salida del array

                while(in.read(data) != -1){
                        out.write(data); // Escribimos en el ByteArrayOutputStream
                }

                // Una vez hecho todo el flujo...
                x_byte = new byte[out.size()];
                x_byte = out.toByteArray();

                in.close();
                out.close();
        }catch (Exception e) {
                }

    } // construirArrayDeBytesDesdeWav()
```

(Volver a la Tabla 2)

XI.3.17 mostrarResultados()

```
public void mostrarResultados(){ // Nos hace la interfaz más amena
        AlertDialog.Builder dialogo = new AlertDialog.Builder(this);

        dialogo.setTitle(R.string.tiempo_de_reverberacion);
        dialogo.setMessage(R.string.quieres_calcular);
        dialogo.setPositiveButton(R.string.si, new OnClickListener() {
           public void onClick(DialogInterface dialog, int which) {
                   //Toast.makeText(getApplicationContext(), R.string.calculando,
Toast.LENGTH_LONG).show();
               dialog.cancel();
               calcularRT(); // Lo hacemos después de que haya grabado la señal de audio al pulsar el
diálogo de alerta.
           }
        });
        dialogo.setNegativeButton(R.string.no, new OnClickListener() {
           public void onClick(DialogInterface dialog, int which) {
               dialog.cancel();
           }
        });

        dialogo.create();
        dialogo.show();
    } // mostrarResultados()
```

(Volver a la Tabla 2)

XI.3.18 calcularRT()

```
public void calcularRT(){ // Calculamos el tiempo de reverberación (es una especie de "main")
        borrarFile(); // Lo primero que hacemos es borrar el fichero, ya que ya lo hemos cargado en un
array y no queremos llenar la memoria de la SD con archivos improductivos

        x_byte=limpiarPrincipioYFinal(x_byte, 0.25); // Limpiamos el principio y la cola de la señal

        double[] x = new double[x_byte.length/2]; // El tamaño es la mitad, ya que se ocupará la mitad de
casillas al redimensionarlo
        x=redimensionarArrayfromBytesToDoubles(x_byte);
        x=interpolar(x); // Ahora queda con el formato de MATLAB (sin normalizar)
        //exportarArrayDeDoublesAFicheroCSVSinTiempo(x, "x");

        x=enventanarArrayDeDoubles(x, 300, 100); // Para cargarnos la parte no útil de la grabación
        //exportarArrayDeDoublesAFicheroCSVSinTiempo(x, "x_enventanado");

        x=hacerArrayDivisiblePor32(x); // Para agilizar cálculos
        //exportarArrayDeDoublesAFicheroCSVSinTiempo(x, "x_divisible");

        x=normalizarArrayDeDoubles(x); // Para gastar menos memoria
        //exportarArrayDeDoublesAFicheroCSVSinTiempo(x, "x_normalizado");

        getSetCoeficientes(); // Sacamos los coeficientes de las clases Butterworth y los metemos en esta
        double[] y_125 = new double[x.length]; // La salida tendrá 6 arrays de doubles (uno por cada
octava)
        double[] y_250 = new double[x.length];
        double[] y_500 = new double[x.length];
        double[] y_1000 = new double[x.length];
        double[] y_2000 = new double[x.length];
        double[] y_4000 = new double[x.length];

        y_125=filtroIIR(x, 125); // Filtramos la señal en bandas de octava
        y_250=filtroIIR(x, 250);
        y_500=filtroIIR(x, 500);
        y_1000=filtroIIR(x, 1000);
        y_2000=filtroIIR(x, 2000);
        y_4000=filtroIIR(x, 4000);
        //exportarArrayDeDoublesAFicheroCSVSinTiempo(y_1000, "y_1000");
        //exportarArrayDeDoublesAFicheroCSVSinTiempo(y_2000, "y_2000");

        y_125=schroeder(y_125); // Hacemos la integral de Schroeder
        y_250=schroeder(y_250);
        y_500=schroeder(y_500);
        y_1000=schroeder(y_1000);
        y_2000=schroeder(y_2000);
        y_4000=schroeder(y_4000);

                y_125=normaLogaritmizar(y_125); // Normalizamos y pasamos a dB
        y_250=normaLogaritmizar(y_250);
        y_500=normaLogaritmizar(y_500);
        y_1000=normaLogaritmizar(y_1000);
        y_2000=normaLogaritmizar(y_2000);
        y_4000=normaLogaritmizar(y_4000);

                posicion_dBStart_125 = posiciones(y_125, dBStart);
                posicion_dBEnd_125 = posiciones(y_125, dBEnd);
                posicion_dBStart_250 = posiciones(y_250, dBStart);
                posicion_dBEnd_250 = posiciones(y_250, dBEnd);
                posicion_dBStart_500 = posiciones(y_500, dBStart);
                posicion_dBEnd_500 = posiciones(y_500, dBEnd);
                posicion_dBStart_1000 = posiciones(y_1000, dBStart);
                posicion_dBEnd_1000 = posiciones(y_1000, dBEnd);
                posicion_dBStart_2000 = posiciones(y_2000, dBStart);
                posicion_dBEnd_2000 = posiciones(y_2000, dBEnd);
                posicion_dBStart_4000 = posiciones(y_4000, dBStart);
                posicion_dBEnd_4000 = posiciones(y_4000, dBEnd);

        reverbTime_125 = (60/(-(double) dBEnd+(double) dBStart)*((double) posicion_dBEnd_125-(double)
posicion_dBStart_125)/(double) RECORDER_SAMPLERATE); // Cálculo del RT
        reverbTime_250 = (60/(-(double) dBEnd+(double) dBStart)*((double) posicion_dBEnd_250-(double)
posicion_dBStart_250)/(double) RECORDER_SAMPLERATE);
```

```
        reverbTime_500 = (60/(-(double) dBEnd+(double) dBStart)*((double) posicion_dBEnd_500-(double)
posicion_dBStart_500)/(double) RECORDER_SAMPLERATE);
        reverbTime_1000 = (60/(-(double) dBEnd+(double) dBStart)*((double) posicion_dBEnd_1000-(double)
posicion_dBStart_1000)/(double) RECORDER_SAMPLERATE);
        reverbTime_2000 = (60/(-(double) dBEnd+(double) dBStart)*((double) posicion_dBEnd_2000-(double)
posicion_dBStart_2000)/(double) RECORDER_SAMPLERATE);
        reverbTime_4000 = (60/(-(double) dBEnd+(double) dBStart)*((double) posicion_dBEnd_4000-(double)
posicion_dBStart_4000)/(double) RECORDER_SAMPLERATE);

        // Metemos el valor de los tiempos de reverberación en los TextView
        textView_t_125.setText(Double.toString(nDecimales(reverbTime_125, 2)));
        textView_t_250.setText(Double.toString(nDecimales(reverbTime_250, 2)));
        textView_t_500.setText(Double.toString(nDecimales(reverbTime_500, 2)));
        textView_t_1000.setText(Double.toString(nDecimales(reverbTime_1000, 2)));
        textView_t_2000.setText(Double.toString(nDecimales(reverbTime_2000, 2)));
        textView_t_4000.setText(Double.toString(nDecimales(reverbTime_4000, 2)));

    } // calcularRT()
```

(Volver a la Tabla 2)

XI.3.19 redimensionarArrayfromBytesToDoubles()

```
public double[] redimensionarArrayfromBytesToDoubles(byte[] entrada){ // Al tener 16 bits (2 bytes) y el
formato "byte" sólo cojer 8 casillas, hay que ir de 2 en 2
        // Conseguimos el doble de información que la que obtendríamos haciendo un "wavread" en MATLAB
        // De hecho, si comparamos los ficheros "csv", el wavread de MATLAB es una interpolación de lo
obtenido aquí con un factor de 2:1
        double[] salida = new double[entrada.length/2];
        int i=0;

        for(int n=0;n<entrada.length;n=n+2){
                salida[i] = (entrada[n]&0xFF) | (entrada[n+1]<<8);
                i=i+1;
        }

        return salida;
        } // redimensionarArrayfromBytesToDoubles()
```

(Volver a la Tabla 2)

XI.3.20 normalizarArrayDeDoubles()

```
public double[] normalizarArrayDeDoubles(double[] entrada){ // Deja en el rango [-1,1] el array de doubles
        double[] salida = new double[entrada.length];

        for(int n=0;n<entrada.length;n=n+1){
                salida[n]=entrada[n]/Math.pow(2, 15);
        }
                return salida;
        } // normalizarArrayDeDoubles()
```

(Volver a la Tabla 2)

XI.3.21 enventanarArrayDeDoubles()

```
public static double[] enventanarArrayDeDoubles(double[] entrada, int ruido_fondo_principio, int
ruido_fondo_final){ // Quita todos lo que sobra por delante y por detrás enventando la información útil de
la señal
        int inicio=0, fin=0; // Tengo que inicializarlos por el bucle

        for(int n=0;n<entrada.length;n=n+1){ // Ceros por la izquierda
                if(entrada[n]>ruido_fondo_principio){
                        inicio=n;
                        break; // salimos del bucle
                }
        }

        for(int n=entrada.length-1;n>0;n=n-1){ // Ceros por la derecha
                if(entrada[n]>ruido_fondo_final){
                        fin=n;
                        break; // salimos del bucle
                }
        }

        int duracion=fin-inicio+1;

        int i=0;
        double[] salida = new double[duracion];
        for (int n=inicio;n<=fin;n=n+1){
                salida[i] = entrada[n];
                i++;
        }

        return salida;
    } //enventanarArrayDeDoubles()
```

(Volver a la Tabla 2)

XI.3.22 getSetCoeficientes()

```
public void getSetCoeficientes(){ // Carga los coeficientes de los filtros

        // Los coeficientes del filtro de Butterworth los calculamos con MATLAB
        // Nos valemos de las clases Butterworth, donde están almacenados todos los coeficientes que nos
interesan
        // De este modo el código, aparte de ser más sencillo, será mucho más rápido
        Butterworth_2 miButterworth_125 = new Butterworth_2();
        Butterworth_4 miButterworth_250 = new Butterworth_4();
        Butterworth_4 miButterworth_500 = new Butterworth_4();
        Butterworth_4 miButterworth_1000 = new Butterworth_4(); // De 1 kHz hacia abajo, de orden 8 se hace
inestable (polos fuera de la circunferencia unidad)
        Butterworth_8 miButterworth_2000 = new Butterworth_8();
        Butterworth_8 miButterworth_4000 = new Butterworth_8();

        // Rellenamos los campos de miButterworth
        miButterworth_125.setB_125();
        miButterworth_125.setA_125();
        miButterworth_250.setB_250();
        miButterworth_250.setA_250();
        miButterworth_500.setB_500();
        miButterworth_500.setA_500();
        miButterworth_1000.setB_1000();
        miButterworth_1000.setA_1000();
        miButterworth_2000.setB_2000();
        miButterworth_2000.setA_2000();
        miButterworth_4000.setB_4000();
        miButterworth_4000.setA_4000();

        // Extraemos el contenido de los campos y lo copiamos a los arrays de esta clase
        B_125 = miButterworth_125.getB_125();
        A_125 = miButterworth_125.getA_125();
        B_250 = miButterworth_250.getB_250();
        A_250 = miButterworth_250.getA_250();
        B_500 = miButterworth_500.getB_500();
        A_500 = miButterworth_500.getA_500();
        B_1000 = miButterworth_1000.getB_1000();
        A_1000 = miButterworth_1000.getA_1000();
        B_2000 = miButterworth_2000.getB_2000();
        A_2000 = miButterworth_2000.getA_2000();
        B_4000 = miButterworth_4000.getB_4000();
        A_4000 = miButterworth_4000.getA_4000();

        // Establecemos el el tamaño de los arays de coeficientes
        tam_125 = miButterworth_125.orden*2+1;
        tam_250 = miButterworth_250.orden*2+1;
        tam_500 = miButterworth_500.orden*2+1;
        tam_1000 = miButterworth_1000.orden*2+1;
        tam_2000 = miButterworth_2000.orden*2+1;
        tam_4000 = miButterworth_4000.orden*2+1;

    } // getSetCoeficientes()
```

(Volver a la Tabla 2)

XI.3.23 exportarArrayDeDoublesAFicheroCSV()

```
public static void exportarArrayDeDoublesAFicheroCSV(double[] entrada, String nombreFile){ // Escribimos un array de doubles en un fichero CSV (añade milisegundos desde 1970)
        // De este modo podemos ver fácilmente lo que vale  cada casilla en MATLAB o EXCEL
        String filepath = Environment.getExternalStorageDirectory().getPath(); // Obtenemos la ruta donde se alojará el fichero de audio
        File file = new File(filepath, CSV_FOLDER);

        if(!file.exists()){
                file.mkdirs(); // Si no existe el directorio, lo creamos
        }

        File outFile = new File(file.getAbsolutePath()+"/"+System.currentTimeMillis()+nombreFile+".csv");

        BufferedWriter buffer = null;
                try {
                        buffer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(outFile)));
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                }

        for (int n=0;n<entrada.length;n++){
            try {
                                buffer.write(""+entrada[n]+"\n");
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
            try {
                                buffer.flush();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
        }

        try {
                        buffer.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
    } // exportarArrayDeDoublesAFicheroCSV()
```

(Volver a la Tabla 2)

XI.3.24 exportarArrayDeBytesAFicheroCSV()

```
public static void exportarArrayDeBytesAFicheroCSV(byte[] entrada, String nombreFile){ // Escribimos un
array de bytes en un fichero CSV (añade milisegundos desde 1970)
        // De este modo podemos ver fácilmente lo que vale  cada casilla en MATLAB o EXCEL
        String filepath = Environment.getExternalStorageDirectory().getPath(); // Obtenemos la ruta donde
se alojará el fichero de audio
        File file = new File(filepath, CSV_FOLDER);

        if(!file.exists()){
                file.mkdirs(); // Si no existe el directorio, lo creamos
        }

        File outFile = new File(file.getAbsolutePath()+"/"+System.currentTimeMillis()+nombreFile+".csv");

        BufferedWriter buffer = null;
                try {
                        buffer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(outFile)));
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                }

        for (int n=0;n<entrada.length;n++){
            try {
                        buffer.write(""+entrada[n]+"\n");
                } catch (IOException e) {
                        e.printStackTrace();
                }
            try {
                        buffer.flush();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        try {
                        buffer.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
    } // exportarArrayDeBytesAFicheroCSV()
```

(Volver a la Tabla 2)

XI.3.25 exportarArrayDeDoublesAFicheroCSVSinTiempo()

```
public static void exportarArrayDeDoublesAFicheroCSVSinTiempo(double[] entrada, String nombreFile){ //
Escribimos un array de doubles en un fichero CSV
        // De este modo podemos ver fácilmente lo que vale  cada casilla en MATLAB o EXCEL
        String filepath = Environment.getExternalStorageDirectory().getPath(); // Obtenemos la ruta donde
se alojará el fichero de audio
        File file = new File(filepath, CSV_FOLDER);

        if(!file.exists()){
                file.mkdirs(); // Si no existe el directorio, lo creamos
        }

        File outFile = new File(file.getAbsolutePath()+"/"+nombreFile+".csv");

        BufferedWriter buffer = null;
                try {
                        buffer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(outFile)));
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                }

        for (int n=0;n<entrada.length;n++){
            try {
                                buffer.write(""+entrada[n]+"\n");
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
            try {
                                buffer.flush();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
        }

        try {
                        buffer.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
    } // exportarArrayDeDoublesAFicheroCSV()
```

(Volver a la Tabla 2)

XI.3.26 exportarArrayDeBytesAFicheroCSVSinTiempo()

```
public static void exportarArrayDeBytesAFicheroCSVSinTiempo(byte[] entrada, String nombreFile){ //
Escribimos un array de bytes en un fichero CSV
        // De este modo podemos ver fácilmente lo que vale  cada casilla en MATLAB o EXCEL
        String filepath = Environment.getExternalStorageDirectory().getPath(); // Obtenemos la ruta donde
se alojará el fichero de audio
        File file = new File(filepath, CSV_FOLDER);

        if(!file.exists()){
                file.mkdirs(); // Si no existe el directorio, lo creamos
        }

        File outFile = new File(file.getAbsolutePath()+"/"+nombreFile+".csv");

        BufferedWriter buffer = null;
                try {
                        buffer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(outFile)));
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                }

        for (int n=0;n<entrada.length;n++){
            try {
                        buffer.write(""+entrada[n]+"\n");
                } catch (IOException e) {
                        e.printStackTrace();
                }
            try {
                        buffer.flush();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        try {
                        buffer.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
    } // exportarArrayDeBytesAFicheroCSVSinTiempo()
```

(Volver a la Tabla 2)

XI.3.27 fxReverb()

```
public double[] fxReverb(double entrada[], int retardo, double atenuacion){ // Le metemos un efecto de
reverb a la señal de entrada
        double[] salida = new double[entrada.length];

                int i;
                double n;
                for(i=0;i<entrada.length;i++){
                        salida[i]=0.0;
                }

                for(i=0;i<=(int)(entrada.length+(1.0-(retardo-1.0)))-1;i++){
                        n=(retardo-1.0)+(double) i;
                        salida[(int) n-1]=entrada[(int) n-1]+atenuacion*entrada[(int) (n-(retardo-2.0))-1];
                }
        return salida;
    } // fxReverb()
```

(Volver a la Tabla 2)

XI.3.28 filtroIIR()

```
public double[] filtroIIR(double entrada[], int octava){ // Filtro de respuesta infinita
        double[] salida = new double[entrada.length];


        int i, n;
        if(octava==125){
                int tam = tam_125;
                double[] buffer = new double[tam];

                for(n=0;n<entrada.length;n=n+1){
                        for(i=0;i<tam-1;i=i+1){
                                buffer[i]=buffer[i+1];
                         }

                     buffer[tam-1]=0;
                     for(i=0;i<tam;i=i+1){
                        buffer[i]=buffer[i]+entrada[n]*B_125[i];
                     }

                     for(i=0;i<tam-1;i=i+1){
                        buffer[i+1]=buffer[i+1]-buffer[0]*A_125[i+1];
                     }
                        salida[n]=buffer[0];
                     }
        }else if(octava==250){
                int tam = tam_250;
                double[] buffer = new double[tam];

                for(n=0;n<entrada.length;n=n+1){
                        for(i=0;i<tam-1;i=i+1){
                                buffer[i]=buffer[i+1];
                         }

                     buffer[tam-1]=0;
                     for(i=0;i<tam;i=i+1){
                        buffer[i]=buffer[i]+entrada[n]*B_250[i];
                     }

                     for(i=0;i<tam-1;i=i+1){
                        buffer[i+1]=buffer[i+1]-buffer[0]*A_250[i+1];
                     }
                        salida[n]=buffer[0];
                     }
        }
        else if(octava==500){
                int tam = tam_500;
                double[] buffer = new double[tam];

                for(n=0;n<entrada.length;n=n+1){
                        for(i=0;i<tam-1;i=i+1){
                                buffer[i]=buffer[i+1];
                         }

                     buffer[tam-1]=0;
                     for(i=0;i<tam;i=i+1){
                        buffer[i]=buffer[i]+entrada[n]*B_500[i];
                     }

                     for(i=0;i<tam-1;i=i+1){
                        buffer[i+1]=buffer[i+1]-buffer[0]*A_500[i+1];
                     }
                        salida[n]=buffer[0];
                     }
        }
        else if(octava==1000){
                int tam = tam_1000;
                double[] buffer = new double[tam];

                for(n=0;n<entrada.length;n=n+1){
                        for(i=0;i<tam-1;i=i+1){
```

```
                        buffer[i]=buffer[i+1];
                    }

                buffer[tam-1]=0;
                for(i=0;i<tam;i=i+1){
                   buffer[i]=buffer[i]+entrada[n]*B_1000[i];
                }

                for(i=0;i<tam-1;i=i+1){
                   buffer[i+1]=buffer[i+1]-buffer[0]*A_1000[i+1];
                }
                   salida[n]=buffer[0];
                }
        }

        else if(octava==2000){
                int tam = tam_2000;
                double[] buffer = new double[tam];

                for(n=0;n<entrada.length;n=n+1){
                        for(i=0;i<tam-1;i=i+1){
                                buffer[i]=buffer[i+1];
                         }

                buffer[tam-1]=0;
                for(i=0;i<tam;i=i+1){
                   buffer[i]=buffer[i]+entrada[n]*B_2000[i];
                }

                for(i=0;i<tam-1;i=i+1){
                   buffer[i+1]=buffer[i+1]-buffer[0]*A_2000[i+1];
                }
                   salida[n]=buffer[0];
                }
        }
        else if(octava==4000){
                int tam = tam_4000;
                double[] buffer = new double[tam];

                for(n=0;n<entrada.length;n=n+1){
                        for(i=0;i<tam-1;i=i+1){
                                buffer[i]=buffer[i+1];
                         }

                buffer[tam-1]=0;
                for(i=0;i<tam;i=i+1){
                   buffer[i]=buffer[i]+entrada[n]*B_4000[i];
                }

                for(i=0;i<tam-1;i=i+1){
                   buffer[i+1]=buffer[i+1]-buffer[0]*A_4000[i+1];
                }
                   salida[n]=buffer[0];
                }
        }

            return salida;
    } // filtroIIR()
```

(Volver a la Tabla 2)

XI.3.29 schroeder()

```
public double[] schroeder(double[] entrada){ // La integral de Schroeder "cumsum(y.^2) en MATLAB"
        double[] salida = new double[entrada.length];

        // 1º: Elevamos al cuadrado el array casilla a casilla
        int n;
        for(n=0;n<entrada.length;n=n+1){
                salida[n] = entrada[n]*entrada[n];
        }

        // 2º: Hacemos la suma acumulada
        int i, k;
        double y_ultima;

        i=0;
        y_ultima=salida[0];
        for(k=0;k<entrada.length-1;k=k+1){
          i=i+1;
          y_ultima+=salida[i];
          salida[i]=y_ultima;
        }

        return salida;
    } // schroeder()
```

(Volver a la Tabla 2)

XI.3.30 normaLogaritmizar()

```
public double[] normaLogaritmizar(double[] entrada){ // Normalizamos y pasamos a dB
        double[] salida = new double[entrada.length];
        double max;
        int i;
        max=entrada[0];
        for(i=0;i<entrada.length-1;i=i+1){
          if(entrada[i+1]>max){
            max=entrada[i+1];
          }
        }

        for(i=0;i<entrada.length;i=i+1){
            salida[i]=10*Math.log10(1-entrada[i]/max);
        }

        return salida;
    } // normaLogaritmizar()
```

(Volver a la Tabla 2)

XI.3.31 posiciones()

```
public int posiciones(double[] entrada, int dB_Start_End){ // Averiguamos las posiciones inicial y final
para hacer el cálculo del RT
        double aux[] = new double[entrada.length];
        int i,j,pos;
        double lugar;
        for(i=0;i<entrada.length;i=i+1){
          aux[i]=Math.abs(entrada[i]-dB_Start_End);
        }
        lugar=aux[0];
        i=-1;
        for(j=0;j<aux.length-1;j=j+1){
          if(aux[j+1]<lugar){
            lugar=aux[j+1];
            i=j;
          }
        }
        pos = i+2;
        return pos;
    } // posiciones()
```

(Volver a la Tabla 2)

XI.3.32 limpiarPrincipioYFinal()

```
public byte[] limpiarPrincipioYFinal(byte[] entrada, double segundos){ // Anulamos lo que nos molesta por
delante y por detrás
        int n;
        double casillas = segundos*RECORDER_SAMPLERATE*2;
        int casillas_int=redondear(casillas);
        for(n=0;n<casillas_int;n=n+1){
                entrada[n]=0;
        }
        for(n=(int) (entrada.length-casillas_int);n<entrada.length;n=n+1){
                entrada[n]=0;
        }
        return entrada;
    } // limpiarPrincipioYFinal()
```

(Volver a la Tabla 2)

XI.3.33 redondear()

```
public static int redondear(double sinRedondear){ // Redondeo de double a entero
        int entero;
        entero = (int) Math.round(sinRedondear);
        return entero;
    } // redondear()
```

(Volver a la Tabla 2)

XI.3.34 nDecimales()

```
public static double nDecimales(double sinRedondear, int cuantosDecimales){ // Para mostrar un determinado
número de decimales
        BigDecimal bd = new BigDecimal(sinRedondear);
        BigDecimal redondeado = bd.setScale(cuantosDecimales, BigDecimal.ROUND_HALF_UP);
        return redondeado.doubleValue();
    } // nDecimales()
```

(Volver a la Tabla 2)

XI.3.35 interpolar()

```
public static double[] interpolar(double[] entrada){ // Interpolación 2:1
        // Este método es para dejar el array igual que hace MATLAB y poder depurar el programa
        double[] salida = new double[entrada.length/2];
        int i=0;
        for(int n=0;n<entrada.length;n=n+2){
                salida[i]=entrada[n];
                i=i+1;
        }
        return salida;
    } // interpolar()
```

(Volver a la Tabla 2)

XI.3.36 hacerArrayDivisiblePor32()

```
public static double[] hacerArrayDivisiblePor32(double[] entrada){ // Redimensionamos la señal de entrada
para que quede con un número de muestras múltiplo de 32
        // Hay que hacer un "floor" para redondear hacia abajo.
        int nuevoTamArray = (int) (Math.floor((double)(entrada.length/32))*32);
        double[] salida = new double[nuevoTamArray];

        for(int n=0;n<salida.length;n++){
                salida[n] = entrada[n];
        }
        return salida;
    } // hacerArrayDivisiblePor32()
```

(Volver a la Tabla 2)

XI.3.37 onCreateOptionsMenu()

```
@Override
        public boolean onCreateOptionsMenu(android.view.Menu menu){ // Mostramos el menu al apretar el
botón físico del dispositivo
                MenuInflater miMenuInflater=getMenuInflater();
                miMenuInflater.inflate(R.menu.menu, menu);
        return true;
        } // onCreateOptionsMenu()
```

(Volver a la Tabla 2)

XI.3.38 onOptionsItemSelected()

```
@Override
        public boolean onOptionsItemSelected(MenuItem item){ // Las 2 opciones del menú
                switch (item.getItemId()) {
        case R.id.item1:
        Toast miToast=Toast.makeText(getApplicationContext(),"AppAcoustiC RT 1.0.0\n\nGabriel Moreno
Ibarra\n\n2012-09-04",Toast.LENGTH_LONG);
        miToast.setGravity(Gravity.CENTER, 0, 0);
        miToast.show();
        break;
        case R.id.item2: finish();
        }
        return true;
        } // onOptionsItemSelected()
```

(Volver a la Tabla 2)