

UNIVERSIDAD DE MÁLAGA  
ESCUELA TÉCNICA SUPERIOR DE  
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESARROLLO DE UNA APLICACIÓN EN  
ANDROID PARA EL CONTROL DE UN  
ESPIRÓMETRO MEDIANTE BLUETOOTH

GRADO EN INGENIERÍA DE  
TECNOLOGÍAS DE TELECOMUNICACIÓN

CARLOS ALBERTO CONTRERAS LÓPEZ  
MÁLAGA, 2014



## **Desarrollo de una Aplicación en Android para el Control de un Espirómetro mediante Bluetooth**

Autor: Carlos Alberto Contreras López

Tutor: Francisco Javier González Cañete

Departamento: Tecnología Electrónica

Titulación: Grado en Ingeniería de Tecnologías de Telecomunicación

Palabras clave: Aplicaciones móviles, Android, Bluetooth, espirómetro, AM1+.

### **Resumen**

Ante la necesidad de servicios médicos a distancia por parte de la población, en los últimos años se está produciendo una pequeña revolución en el ámbito de la medicina. Cada vez más se utilizan las tecnologías de la información y las comunicaciones para proporcionar asistencia médica a pacientes sin que éstos tengan que encontrarse personalmente con los profesionales de la medicina.

En otro ámbito, la implantación de *smartphones* (terminales móviles con mayores prestaciones y numerosas posibilidades de conectividad) en nuestra sociedad es casi total.

En este contexto se plantea este Trabajo Fin de Grado, cuyo objetivo es desarrollar una aplicación para dispositivos Android que permita conectarse mediante Bluetooth al espirómetro AM1+ de la marca JAEGER y realizar operaciones de consulta de los resultados de la espirometría, visualización de éstos en gráficos, borrado de la memoria del espirómetro, etc. La aplicación se ha desarrollado en lenguaje nativo, esto es, usando Java; y los interfaces de usuario se han construido mediante el lenguaje XML.



## **Development of an Android Application to Control a Spirometer via Bluetooth**

Author: Carlos Alberto Contreras López

Supervisor: Francisco Javier González Cañete

Department: Tecnología Electrónica

Degree: Grado en Ingeniería de Tecnologías de Telecomunicación

Keywords: Mobile applications, Android, Bluetooth, spirometer, AM1+.

### **Abstract**

In the last few years, it is happening a small revolution in the field of Medicine because of the need for medical services at a distance by the population. Communications and information technologies are used to provide medical care to patients without having to personally meet with the medical professionals.

On the other hand, the introduction of smartphones (mobile terminals with high performance and lots of connectivity) in our society is almost complete.

This project arises in this context. Its objective is to develop an application for Android devices that allows to connect via Bluetooth to the JAEGER brand spirometer AM1, query the results of the spirometry, display these data in graphics, erase the memory of the spirometer, etc. The application has been developed in native language, namely, using Java; and the user interfaces have been built using the XML language.



A mi familia y amigos.

*Carlos.*





# Contenido

<b>Acrónimos</b>	<b>VII</b>
<b>Índice de Figuras</b>	<b>IX</b>
<b>Índice de Tablas</b>	<b>XI</b>
<b>1 Introducción</b>	<b>1</b>
<b>2 Tecnologías y Herramientas Empleadas</b>	<b>5</b>
2.1 Android . . . . .	5
2.2 Bluetooth . . . . .	8
2.3 XML . . . . .	9
2.4 Eclipse . . . . .	10
2.5 JAEGER Asthma Monitor AM1+ . . . . .	12
<b>3 Interfaces de Usuario</b>	<b>17</b>
3.1 <i>activity_main.xml</i> . . . . .	17
3.2 <i>activity_grafica.xml</i> . . . . .	19
3.3 <i>mostrar_datos.xml</i> . . . . .	20
3.4 <i>textview_datos.xml</i> . . . . .	21
3.5 <i>textview_encabezado.xml</i> . . . . .	22

<b>4</b>	<b>Lógica de la Aplicación</b>	<b>23</b>
4.1	Clase <i>MainActivity</i>	23
4.1.1	Método <i>onCreate(Bundle savedInstanceState)</i>	25
4.1.2	<i>Handler</i>	25
4.1.3	Código del Atributo <i>onClick</i> de <i>BotonAcercaDe</i>	27
4.1.4	Código del Atributo <i>onClick</i> de <i>BotonConectar</i>	28
4.1.5	Código del Atributo <i>onClick</i> de <i>BotonVersion</i>	28
4.1.6	Método <i>MostrarVersion(String cadena)</i>	29
4.1.7	Código del Atributo <i>onClick</i> de <i>BotonPaciente</i>	29
4.1.8	Código del Atributo <i>onClick</i> de <i>BotonMediciones</i>	30
4.1.9	Método <i>MostrarDatos(String[] datos)</i>	30
4.1.10	Código del Atributo <i>onClick</i> de <i>BotonBorrar</i>	32
4.1.11	Código del Atributo <i>onClick</i> de <i>BotonSalir</i>	33
4.1.12	Método <i>Botones(boolean estado)</i>	34
4.1.13	Método <i>ActivarBT()</i>	34
4.1.14	Método <i>onActivityResult(int requestCode, int resultCode, Intent data)</i>	35
4.1.15	Método <i>Receptor()</i>	35
4.1.16	Método <i>onDestroy()</i>	35
4.1.17	Método <i>onCreateOptionsMenu(Menu menu)</i>	35
4.1.18	Método <i>onOptionsItemSelected(MenuItem item)</i>	35
4.1.19	Método <i>onSaveInstanceState(Bundle EstadoGuardado)</i>	36
4.1.20	Método <i>onRestoreInstanceState(Bundle EstadoGuardado)</i>	36
4.2	Clase <i>ConexionBT</i>	37
4.2.1	Constructor <i>ConexionBT()</i>	38
4.2.2	Método <i>BluetoothDevice DispositivoEmparejado(String nombre_disp)</i>	38

4.2.3	Método <i>EstablecerConexion(String nombre_disp, String uuidString)</i> . . . . .	39
4.2.4	Método <i>Enviar(byte[] datos)</i> . . . . .	40
4.2.5	Método <i>Enviar(String datos)</i> . . . . .	40
4.2.6	Método <i>int Recibir(byte[] buffer)</i> . . . . .	40
4.2.7	Método <i>Cerrar()</i> . . . . .	40
4.3	Hebra <i>HiloInicio</i> . . . . .	40
4.3.1	Constructor <i>HiloInicio(ConexionBT conexion, Handler handler, String disp_emparejado, String uuidString)</i> . . . . .	41
4.3.2	Método <i>run()</i> . . . . .	41
4.4	Hebra <i>HiloFechayHora</i> . . . . .	43
4.4.1	Constructor <i>HiloFechayHora(ConexionBT conexion, Handler handler)</i> . . . . .	43
4.4.2	Método <i>run()</i> . . . . .	44
4.5	Hebra <i>HiloVersion</i> . . . . .	45
4.5.1	Constructor <i>HiloVersion(ConexionBT conexion, Handler handler)</i> . . . . .	45
4.5.2	Método <i>run()</i> . . . . .	45
4.6	Hebra <i>HiloPaciente</i> . . . . .	46
4.6.1	Constructor <i>HiloPaciente(ConexionBT conexion, Handler handler, String nombre)</i> . . . . .	47
4.6.2	Método <i>run()</i> . . . . .	47
4.7	Hebra <i>HiloMediciones</i> . . . . .	48
4.7.1	Constructor <i>HiloMediciones(ConexionBT conexion, Handler handler)</i> . . . . .	49
4.7.2	Método <i>Ordenar(String buffer, int NumMed, int bloque)</i> . . . . .	49
4.7.3	Método <i>run()</i> . . . . .	50

4.8	Hebra <i>HiloBorrar</i> . . . . .	53
4.8.1	Constructor <i>HiloBorrar(ConexionBT conexion, Handler handler)</i> . . . . .	53
4.8.2	Método <i>run()</i> . . . . .	54
4.9	Hebra <i>HiloSalir</i> . . . . .	54
4.9.1	Constructor <i>HiloSalir(ConexionBT conexion, Handler handler)</i> . . . . .	55
4.9.2	Método <i>run()</i> . . . . .	55
4.10	Clase <i>ActivityGrafica</i> . . . . .	55
4.10.1	Método <i>onCreate(Bundle savedInstanceState)</i> . . . . .	56
4.11	Fichero <i>AndroidManifest.xml</i> . . . . .	57
<b>5</b>	<b>Plan de Pruebas</b>	<b>59</b>
	<b>Conclusiones y Líneas Futuras</b>	<b>63</b>
<b>A</b>	<b>Manual de Usuario</b>	<b>65</b>
	<b>Bibliografía</b>	<b>71</b>

# Acrónimos

<b>ACK</b>	ACKnowledgement
<b>API</b>	Application Programming Interface
<b>AVD</b>	Android Virtual Devices
<b>BT</b>	BlueTooth
<b>ECJ</b>	Eclipse Compiler for Java
<b>FEF</b>	Forced Expiratory Flow
<b>FEP</b>	Flujo Espiratorio Pico
<b>FVC</b>	Forced Vital Capacity
<b>FEV</b>	Forced Expiratory Volume
<b>HID</b>	Human Interface Device
<b>HTML</b>	Hyper Text Mark-up Language
<b>ID</b>	IDentifier
<b>IDE</b>	Integrated Development Environment
<b>iOS</b>	iPhone Operating System

<b>OHA</b>	Open Handset Alliance
<b>RFCOMM</b>	Radio Frequency COMMunication
<b>SDK</b>	Software Development Kit
<b>SGML</b>	Standard Generalized Markup Language
<b>UI</b>	User Interface
<b>USB</b>	Universal Serial Bus
<b>UUID</b>	Universally Unique IDentifier
<b>W3C</b>	World Wide Web Consortium
<b>XHTML</b>	eXtensible Hyper Text Mark-up Language
<b>XML</b>	eXtensible Markup Language

# Índice de Figuras

2.1	Uso de las distintas versiones de Android. . . . .	6
2.2	Entorno de programación Eclipse. . . . .	11
2.3	La herramienta LogCat. . . . .	12
2.4	El espirómetro AM1+. . . . .	12
3.1	Pantalla de inicio de la aplicación. . . . .	19
3.2	Gráfica con los valores del parámetro PEF. . . . .	20
3.3	Resultados de la espirometría. . . . .	21
4.1	Diagrama de clases del proyecto. . . . .	24
4.2	Diagrama de la clase <i>MainActivity</i> . . . . .	25
4.3	Información mostrada al pulsar <i>BotonAcercaDe</i> . . . . .	27
4.4	Secuencia cuando el usuario pulsa <i>BotonVersion</i> . . . . .	28
4.5	Información mostrada al pulsar <i>BotonVersion</i> . . . . .	29
4.6	Secuencia cuando el usuario pulsa <i>BotonBorrar</i> y confirma. . . . .	32
4.7	Secuencia cuando el usuario pulsa <i>BotonSalir</i> y confirma. . . . .	33
4.8	Petición de activación de Bluetooth. . . . .	34
4.9	Diagrama de la clase <i>ConexionBT</i> . . . . .	37
4.10	Diagrama de la clase <i>HiloInicio</i> . . . . .	41
4.11	Secuencia cuando el usuario pulsa <i>BotonConectar</i> . . . . .	42
4.12	Diagrama de la clase <i>HiloFechayHora</i> . . . . .	43

4.13 Diagrama de la clase <i>HiloVersion</i> . . . . .	45
4.14 Diagrama de la clase <i>HiloPaciente</i> . . . . .	46
4.15 Cuadro de diálogo mostrado al pulsar <i>BotonPaciente</i> . . . . .	48
4.16 Diagrama de la clase <i>HiloMediciones</i> . . . . .	48
4.17 Secuencia cuando el usuario pulsa <i>BotonMediciones</i> . . . . .	51
4.18 Diagrama de la clase <i>HiloBorrar</i> . . . . .	53
4.19 Diagrama de la clase <i>HiloSalir</i> . . . . .	55
4.20 Diagrama de la clase <i>ActivityGrafica</i> . . . . .	56
4.21 Gráfica con los valores del parámetro PEF. . . . .	57
A.1 Bluetooth del espirómetro activado. . . . .	65
A.2 Botones habilitados. . . . .	66
A.3 Información acerca de la versión del espirómetro. . . . .	66
A.4 Cuadro de diálogo para cambiar el nombre del paciente. . . . .	67
A.5 Datos de las mediciones. . . . .	67
A.6 Gráfica con los valores del parámetro PEF. . . . .	68
A.7 Confirmación para borrar la memoria del espirómetro. . . . .	68
A.8 Confirmación para salir de la aplicación. . . . .	69
A.9 Pantalla inicial de la aplicación. . . . .	69



# Índice de Tablas

2.1	Versiones del sistema operativo Android. . . . .	7
2.2	Parámetros de la espirometría. . . . .	13
2.3	Comandos del espirómetro. . . . .	13
5.1	Resultados de las pruebas con varios dispositivos. . . . .	61



# Capítulo 1

## Introducción

El progresivo envejecimiento de la población mundial, el avance frente a enfermedades que antes eran mortales y ahora se han convertido en crónicas gracias a los continuos avances médicos, unido al traslado de algunos servicios desde los centros especializados a centros de atención primaria e incluso hogares, está generando por parte de la sociedad nuevas formas de cuidado de la salud. Es aquí donde las tecnologías de la información y las comunicaciones están siendo utilizadas para dar soporte una amplia gama de modelos de cuidado sanitario que aspiran a facilitar el día a día de personas con enfermedades en su mayoría crónicas y que necesitan servicios de monitorización o tratamiento domiciliario. [1]. Estas técnicas se denominan telemedicina.

Pero, ¿qué es exactamente la telemedicina? Es cualquier acto médico realizado sin contacto físico directo entre el profesional y el paciente, por medio de algún sistema telemático. En otras palabras, la telemedicina utiliza las tecnologías de la información y las telecomunicaciones para proporcionar o soportar la asistencia médica, sin importar la distancia que separa a los que ofrecen y hacen uso del servicio. [2]. Nótese la gran importancia de esto por ejemplo en núcleos de población rurales, en los que un paciente puede encontrarse incluso a centenares de kilómetros de un

centro médico especializado.

Los servicios de telemedicina tienen además varias ventajas dignas de mencionar, que ofrecen una oportunidad incomparable para revolucionar una parte importante de la medicina: la elevada calidad de los servicios sociales y de salud ofrecidos a los ciudadanos, la inmediatez y facilidad de acceso a esos servicios por parte de los pacientes y su alta eficiencia, tan demanda por los gobiernos en tiempos de inestabilidad económica.

La telemedicina es aplicable a múltiples áreas de la salud. Una de ellas es la espirometría. La espirometría es una prueba básica para el estudio de la función pulmonar y su realización es necesaria para la evaluación y el seguimiento de las enfermedades respiratorias, como por ejemplo asma o fibrosis quística [3]. Su utilidad trasciende el ámbito de la neumología, y en los últimos años se está incorporando progresivamente en atención primaria y otras disciplinas médicas.

En otro ámbito, el uso de *smartphones*, teléfonos móviles que pueden comunicarse a través de Wi-Fi, Bluetooth, conexión a Internet y permiten a los usuarios la instalación de programas, ha crecido de tal modo que se ha convertido en un elemento más de nuestra vida cotidiana. Además, según el informe Spain Digital Future in Focus de comScore [4], España es el país líder en Europa en uso de *smartphones* con un 66 % de penetración, mientras que la media en EU5 (Inglaterra, Francia, Italia, Alemania y España) es del 57 %.

Existen distintos sistemas operativos para estos dispositivos. El más extendido es *Android* que, según el último estudio de Kantar Worldpanel ComTech sobre cuotas de mercado de los sistemas operativos ha alcanzado el 64,6 % de la cuota de *smartphones* a nivel mundial, convirtiéndose así en la primera plataforma de venta de teléfonos inteligentes. En cuanto a Europa, *Android* ha acaparado el 70,4 % de la cuota de mercado, *iOS* (iPhone Operating System) un 17,8 % y *Windows Phone* un 6,8 %. [4]

En este contexto se propone el presente Trabajo Fin de Grado: el desarrollo de una aplicación en *Android* para interactuar con un espirómetro, en concreto, con el modelo *Asthma Monitor AM1+* de la marca JAEGER, que cuenta con un módulo Bluetooth. El cometido de la aplicación es ofrecer al usuario funciones que no satisface el espirómetro con su display y botones, tales como visualizar el valor de todos los parámetros para cada medición, observar la evolución de las medidas en un gráfico, borrar la memoria del espirómetro, cambiar el nombre del paciente, etc.

El presente documento está estructurado en diversos capítulos para poder explicar con detalle todas las partes del trabajo:

- 1. Introducción.** Es el presente capítulo, en el que se hace una somera descripción y justificación del trabajo, así como de los objetivos a conseguir.
- 2. Tecnologías y herramientas empleadas.** En este capítulo se detallan las herramientas software y hardware utilizadas en el desarrollo del Trabajo Fin de Grado.
- 3. Interfaces de usuario.** Aquí se describen las distintas interfaces de pantalla construidas para la interacción de la aplicación desarrollada con el usuario.
- 4. Lógica de la aplicación.** En este capítulo se describe en profundidad toda la lógica implementada en el trabajo.
- 5. Plan de pruebas.** En este apartado se detallan cada una de las pruebas realizadas al sistema para comprobar su correcto funcionamiento.
- 6. Conclusiones y líneas futuras.** En el último capítulo se hace un balance del Trabajo Fin de Grado, así como el estudio de posibles mejoras o ampliaciones.



## Capítulo 2

# Tecnologías y Herramientas

## Empleadas

En este capítulo se detallan las tecnologías que se han empleado para la realización de este trabajo, así como las herramientas software utilizadas y la labor para la que fueron utilizadas.

### 2.1. Android

Al contrario de lo que se suele pensar, Google no creó Android. La empresa Android Incorporation (la cual ya no existe como tal, sino que se trata de una parte de la empresa Google) fue fundada en Palo Alto, California, en Octubre del 2003 por Andy Rubin (cofundador de Danger Incorporation, una empresa que trabajaba exclusivamente en plataformas, software, diseño y servicios para dispositivos móviles), Rich Miner (cofundador de Wildfire Communications Incorporation), Nick Sears (fue vicepresidente de T-Mobile) y Chris White (encabezó el desarrollo del diseño y la interfaz en Web TV). En el inicio de Android Incorporation sólo se conocía que trabajaban en el desarrollo de software para dispositivos móviles [5].

Google adquirió Android Incorporation en verano del año 2005, incorporándose Andy Rubin, Rich Miner y Chris White a su plantilla. No se conoce mucho más de la adquisición de Android Incorporation por parte de Google, pero se rumoreó con que Google estaba planeando entrar en el mercado de la telefonía móvil [6].

En Noviembre del 2007 se publicó el primer SDK de Android. Un SDK (Software Development Kit) es un kit de desarrollo software que permite hacer nuevas aplicaciones. En ese mismo mes, la OHA (Open Handset Alliance), un consorcio de varias compañías (Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology-Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel, T-Mobile y Texas Instruments) surgió con el propósito de desarrollar estándares abiertos para dispositivos móviles. Al mismo tiempo, la OHA sacó a la luz su primer producto, Android, un sistema operativo construido sobre el kernel de Linux en su versión 2.6. En diciembre del año 2008, se unieron 14 miembros nuevos (ARM Holdings, Asustek Computer Incorporation, Garmin Ltd, Huawei Technologies, Packet Video, Atheros Communications, Vodafone, Sony Ericsson, Toshiba Corporation).

Desde entonces, han surgido varias versiones del S.O. Android, que arreglan errores y añaden nuevas funcionalidades con respecto a las versiones anteriores [7].

En la figura 2.1 se muestra el uso actual de las versiones más usadas.

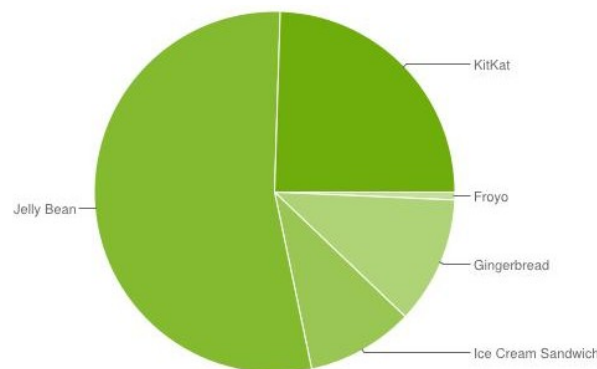


Figura 2.1: Uso de las distintas versiones de Android.



Cabe destacar que casi uno de cada cuatro terminales Android usa la última versión: KitKat.

En la tabla 2.1 se puede ver un listado de todas las versiones.

<b>Versión</b>	<b>Lanzamiento</b>	<b>Nivel API</b>
Android 1.0	Septiembre de 2008	API 1
Android 1.1	Febrero de 2009	API 2
Android 1.5, Cupcake	Abril de 2009	API 3
Android 1.6, Donut	Septiembre de 2009	API 4
Android 2.0, Éclair	Octubre de 2009	API 5
Android 2.1	Enero de 2010	API 7
Android 2.2, Froyo	Mayo de 2010	API 8
Android 2.3, Gingerbread	Diciembre de 2010	API 9
Android 3.0, Honeycomb	Febrero de 2011	API 11
Android 3.1	Mayo de 2011	API 12
Android 3.2	Julio de 2011	API 13
Android 4.0, Ice Cream Sandwich	Octubre de 2011	API 14
Android 4.0.3	Diciembre de 2011	API 15
Android 4.1, Jelly Bean	Julio de 2012	API 16
Android 4.2	Noviembre de 2012	API 17
Android 4.3	Julio de 2013	API 18
Android 4.4, KitKat	Octubre de 2013	API 19

Tabla 2.1: Versiones del sistema operativo Android.

## 2.2. Bluetooth

Bluetooth es la tecnología que se ha usado para conectar el dispositivo Android donde se ejecuta la aplicación con el espirómetro.

Es una tecnología de radio de corto alcance, que permite conectividad inalámbrica entre dispositivos remotos. Se diseñó pensando básicamente en tres objetivos: pequeño tamaño, mínimo consumo y bajo precio.

Opera en la banda libre de radio ISM (en inglés, Industrial, Scientific and Medical, banda industrial científico-médica) a 2.4 GHz. Su máxima velocidad de transmisión de datos es de 1 Mbps. El rango de alcance Bluetooth depende de la potencia empleada en la transmisión. La mayor parte de los dispositivos que usan Bluetooth transmiten con una potencia nominal de salida de 0 dBm, lo que permite un alcance de unos 10 metros en un ambiente libre de obstáculos.

Desde su aparición hasta la actualidad se ha ido mejorando el estándar (versiones 1.0, 1.1, 2.0, 2.1, 3.0 y 4.0) persiguiendo como objetivos la reducción del consumo y el aumento de la velocidad de transferencia.

El desarrollo de aplicaciones Bluetooth en Android es posible gracias al paquete *android.bluetooth* que contiene unas APIs que permiten centrarse en el desarrollo, en vez de los detalles de bajo nivel de Bluetooth, escondiendo de esta forma la complejidad del protocolo. [8]

Haciendo uso de estas APIs se ha conseguido lo siguiente:

- Acceder al adaptador local del dispositivo para obtener los dispositivos emparejados.
- Establecer un canal de comunicación RFCOMM (Radio Frequency Communication).
- Usar dicha conexión para enviar y recibir datos.

- Registrar cambios en el Bluetooth del dispositivo donde se ejecuta la aplicación.
- Solicitar petición al usuario para que active Bluetooth.

## 2.3. XML

XML (eXtensible Markup Language, Lenguaje de Marcas Extensible en castellano) es el lenguaje utilizado para definir las interfaces de usuario de la aplicación.

XML no es un lenguaje de marcas, sino un metalenguaje, es decir, define las reglas generales que debe cumplir un lenguaje de marcas y la manera de definir un lenguaje de marcas. [9]

XML fue creado por el W3C<sup>1</sup> (World Wide Web Consortium) a finales de los 90. El W3C se fundó en 1994 para tutelar el crecimiento y organización de la web. Su primer trabajo fue normalizar HTML (Hyper Text Mark-up Language), el lenguaje de marcas con el que se escriben las páginas web. Al crecer el uso de la web, crecieron las presiones para ampliar HTML. El W3C decidió que la solución no era ampliar HTML, sino crear unas reglas para que cualquiera pudiera crear lenguajes de marcas adecuados a sus necesidades, pero manteniendo unas estructuras y sintaxis comunes que permitieran compatibilizarlos y tratarlos con las mismas herramientas. Ese conjunto de reglas es XML, cuya primera versión se publicó en 1998.

Lógicamente, HTML no cumple las normas de XML ya que HTML es anterior a XML. El creador de HTML, Tim Berners-Lee, se basó en SGML, otro conjunto de reglas para la creación de lenguajes de marcas creado en los años 80 y más complejo que XML. Una vez creado XML, el W3C aprobó en el año 2000 XHTML (eXten-

---

<sup>1</sup>W3C es una comunidad internacional donde las organizaciones miembro, personal a tiempo completo y el público en general trabajan conjuntamente para desarrollar estándares Web.

sible Hyper Text Mark-up Language), una versión de HTML que sí que cumple las reglas de XML. El W3C pretendió sin éxito que HTML dejara de utilizarse y sólo se utilizara XHTML. Al no conseguirlo, el W3C decidió retomar el desarrollo de HTML (incluyendo en él una versión XHTML).

Por su parte, el éxito de XML ha sido enorme y cada vez es más utilizado como sistema de intercambio y almacenamiento de información. El W3C ha desarrollado alrededor de XML numerosas tecnologías para sacar provecho de él.

## 2.4. Eclipse

Eclipse es un entorno de desarrollo multiplataforma basado en Java, de código abierto y extensible mediante plugins. Es el *IDE* (Integrated Development Environment) oficial del proyecto Android. Un IDE es un programa compuesto por un conjunto de herramientas para desarrolladores de software que cuenta como elementos básicos, con un editor de código, un compilador/intérprete y un depurador. Eclipse da soporte a multitud de lenguajes, entre los que se encuentran C/C++, Cobol, Fortran, PHP o Python. Esta plataforma ha sido usada para desarrollar otros IDE como el de Java (JDT, Java Development Toolkit) y el compilador ECJ (Eclipse Compiler for Java), que ya se encuentra integrado con Eclipse [10].

Una gran parte de la programación inicial de Eclipse fue realizada por IBM como sucesor de la familia de herramientas para Visual Age. En Noviembre del 2001, se formó un consorcio para el desarrollo del proyecto Eclipse como código abierto, con IBM y Borland a la cabeza. A finales del 2003, este consorcio había crecido hasta los 80 miembros. En Febrero del 2004, se anunció la reorganización de Eclipse a una fundación sin ánimo de lucro [11]. La figura 2.2 muestra una imagen del entorno Eclipse.

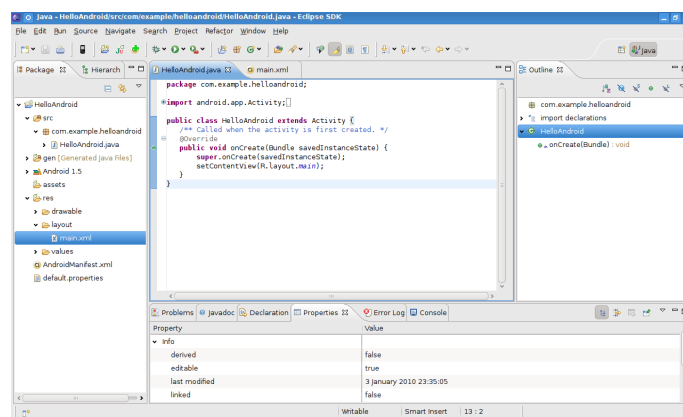


Figura 2.2: Entorno de programación Eclipse.

Para poder desarrollar aplicaciones para el sistema operativo Android con Eclipse es necesaria la instalación del ADT Plugin (Android Development Tools). Dicho plugin extiende las capacidades de Eclipse para desarrollar nuevos proyectos Android, crear interfaces de usuario de las aplicaciones, añadir paquetes basados en la API de Android y depurar las aplicaciones usando el SDK de Android. También permite exportar los archivos de aplicación con extensión .apk con el objeto de distribuir las aplicaciones desarrolladas.

Para poder comprobar el correcto funcionamiento de las aplicaciones el entorno dispone del simulador AVD (Android Virtual Devices), disponible en las herramientas del SDK de Android. Sin embargo, no es posible simular aplicaciones que hacen uso de Bluetooth. Así pues, las pruebas se realizan sobre un dispositivo físico conectado al ordenador a través de un cable USB.

Para la depuración, se hace uso de la herramienta *LogCat*, que consiste en la introducción de mensajes durante el código de la aplicación, que después se capturan y se muestran por pantalla durante la ejecución. La figura 2.3 muestra una imagen de dicha utilidad.

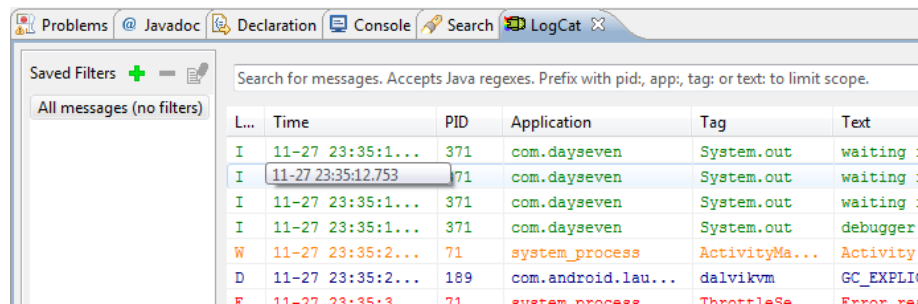


Figura 2.3: La herramienta LogCat.

## 2.5. JAEGER Asthma Monitor AM1+

Se trata del modelo de espirómetro con el que se conecta mediante Bluetooth la aplicación desarrollada. En la figura 2.4 puede observarse una imagen del dispositivo.



Figura 2.4: El espirómetro AM1+.

Por cada medición, el espirómetro *AM1* almacena siete parámetros relativos a la espirometría [12]: FVC (Forced Vital Capacity), FEP (Flujo Espiratorio Pico), FEF (Forced Expiratory Flow), FEF75 (Forced Expiratory Flow 75 %), FEF50 (Forced Expiratory Flow 50 %), FEF25(Forced Expiratory Flow 25 %) y FEF7525 (Forced Expiratory Flow promedio de FEF75 Y FEF25). Dichos parámetros se detallan en la tabla 2.2.

<b>Parámetro (unidad)</b>	<b>Explicación</b>
FVC (l)	Volumen total de aire expulsado
FEP (l/min)	Flujo máximo conseguido
FEV1 (l)	Volumen máximo en el primer segundo
FEF75 (l/s)	Flujo al 75 % de FVC
FEF50 (l/s)	Flujo al 50 % de FVC
FEF25 (l/s)	Flujo al 25 % de FVC
FEF75/25 (l/s)	Flujo medio entre FEF25 y FEF75

Tabla 2.2: Parámetros de la espirometría.

El espirómetro tiene definidos algunos comandos para comunicarse con él. En la tabla 2.3 se detallan los que se han usado en la aplicación. <CR>se refiere a retorno de carro y <LF>a fin de línea.

<b>Comando</b>	<b>Sintaxis</b>	<b>Respuesta del AM1</b>
Inicio comunicación	ENQ(0x05)	ACK(0x06)
Establecer fecha	Ddd.mm.aa<CR>	ACK
Establecer hora	Thh.mm<CR>	ACK
Consultar versión	v<CR>	ACK + v[version]<CR><LF>
Cambiar paciente	I[paciente]<CR>	ACK
Obtener bloque	B[numero]<CR>	ACK + [bloque]<CR><LF>
Obt. núm. medidas	C<CR>	ACK + B[med]:[keys]<CR><LF>
Borrar memoria	L<CR>	ACK
Apagar	k<CR>	ACK

Tabla 2.3: Comandos del espirómetro.

A continuación se ponen algunos ejemplos de la comunicación entre la aplicación

y el espirómetro.

**Inicio de la comunicación.** Enviado al AM1: ENQ(0x05). Respuesta: ACK(0x06).

**Establecer fecha.** Enviado al AM1: D15.09.14<CR>. Respuesta: ACK.

**Establecer hora.** Enviado al AM1: T14.34<CR>. Respuesta: ACK.

**Consultar versión.** Enviado al AM1: v<CR>. Respuesta: ACK seguido de  
v730:B201043<CR><LF>.

**Cambiar el nombre del paciente.** Enviado al AM1: ICarlos<CR>. Respuesta:  
ACK.

**Obtener el número de medidas.** Enviado al AM1: C<CR>. Respuesta: ACK  
seguido de B0005:0001<CR><LF>. En este ejemplo hay cinco medidas y  
una clave almacenadas (la aplicación desarrollada no hace uso de las claves).  
El espirómetro envía las medidas bloque a bloque. Como máximo, un bloque  
contiene cien medidas.

**Obtener un bloque de datos.** Enviado al AM1: B01<CR>. Respuesta: ACK se-  
guido del bloque de datos número uno. Cada bloque tiene la siguiente estruc-  
tura:

```
[longitud línea]3600[nombre del paciente]<CR><LF>
[longitud línea]7900:[valor 1 FVC]:[valor 2 FVC]:...:[valor N FVC]<CR><LF>
[longitud línea]7901:[valor 1 PEF]:[valor 2 PEF]:...:[valor N PEF]<CR><LF>
[longitud línea]7902:[valor 1 FEV1]:[valor 2 FEV1]:...:[valor N FEV1]<CR><LF>
[longitud línea]7903:[valor 1 FEF75]:[valor 2 FEF75]:...:[valor N FEF75]<CR><LF>
[longitud línea]7904:[valor 1 FEF50]:[valor 2 FEF50]:...:[valor N FEF50]<CR><LF>
[longitud línea]7905:[valor 1 FEF25]:[valor 2 FEF25]:...:[valor N FEF25]<CR><LF>
[longitud línea]7906:[valor 1 FEF7525]:[valor 2 FEF7525]:...:[valor N FEF7525]
```



<CR><LF>

[longitud línea]7908:[valor 1 fecha]:[valor 2 fecha]:...:[valor N fecha]<CR><LF>

[longitud línea]7909:[valor 1 hora]:[valor 2 hora]:...:[valor N hora]<CR><LF>

[longitud línea]7911:[valor 1 key]:[valor 2 key]:...:[valor N key]<CR><LF>

0098540<CR><LF>

**Borrar memoria.** Enviado al AM1: L<CR>. Respuesta: ACK

**Apagar.** Enviado al AM1: k<CR>. Respuesta: ACK

Todos los datos enviados por el AM1 tienen formato carácter. Se ha de mencionar que, aunque la documentación del espirómetro dice que cada valor de parámetros ocupa cuatro caracteres, se ha comprobado que los valores de los parámetros FEF75, FEF50, FEF25 y FEF7525 tienen un carácter ‘0’ previo a los cuatro mencionados.



# Capítulo 3

## Interfaces de Usuario

En este capítulo se describe el diseño de las pantallas de la aplicación. En *Android*, estas interfaces se definen en ficheros *XML*.

### 3.1. *activity\_main.xml*

Esta interfaz es la pantalla de la *MainActivity*. Se asocia a ella en el método *onCreate* con *setContentView(R.layout.activity\_main)*.

En primer lugar se define un *layout*<sup>1</sup> de tipo *RelativeLayout*<sup>2</sup> que contendrá el resto de controles de la interfaz. Se definen su altura y anchura ajustadas al contenido y se centra horizontal y verticalmente. Dentro de él se definen los siguientes controles:

---

<sup>1</sup>Un layout es un elemento no visual destinado a controlar la distribución, posición y dimensiones de los controles (botones, etiquetas, cuadros de texto, etc) que se insertan en su interior.

<sup>2</sup>Este tipo de layout permite especificar la posición de cada elemento de forma relativa a su elemento padre o a cualquier otro elemento incluido en el propio layout.

- Un botón con *id BotonAcercaDe*<sup>3</sup>, texto “Acerca de”, tamaño de texto 10 dp<sup>4</sup>, altura 35 dp, ancho ajustado al contenido y estilo de texto negrita. Se posiciona al principio del *layout* y con su borde derecho alineado con el borde derecho del *layout layout\_centrado1*, definido más adelante. Su atributo *onClick* se establece como “AcercaDe”. Este nombre se utiliza en la parte lógica para establecer las acciones que se llevan a cabo cuando el usuario pulsa el botón.
- Un botón centrado horizontalmente, con *id BotonConectar*, texto “Conectar” y atributo *onClick* idéntico al *id*. Se sitúa debajo de *BotonAcercaDe*.
- Un *LinearLayout*<sup>5</sup> llamado *layout\_centrado1*, con orientación horizontal, ancho y alto ajustado a su contenido y posición debajo de *BotonConectar*. Este *layout* contiene dos botones, *BotonVersion* y *BotonPaciente*, cuyos atributos texto son “Consultar Versión AM1” y “Cambiar Paciente”, respectivamente; y sus atributos *onClick* “Paciente” y “Version”.
- Otro *LinearLayout*, llamado *layout\_centrado2*, debajo del anterior, también con orientación horizontal, y que contiene dos botones: *BotonMediciones*, con texto “Obtener Mediciones” y atributo *onClick*, “Mediciones”; y *BotonBorrar*, con texto y atributo *onClick* “Borrar Datos”.
- Un botón centrado horizontalmente, cuyo *id* es *BotonSalir*, texto “Salir” y atributo *onClick* del mismo nombre. Se sitúa debajo de *layout\_centrado2*.

Puesto que todos los botones excepto *BotonAcercaDe* deben tener la misma apariencia, se define un estilo llamado *EstiloBoton* en el fichero *styles.xml* y se

---

<sup>3</sup>Se añade “@+id/” delante de “BotonAcercaDe” para que al compilar se genere automáticamente una nueva constante para este control.

<sup>4</sup>Esta unidad es equivalente a un píxel en una pantalla con una densidad de 160 puntos por pulgada.

<sup>5</sup>Este *layout* apila uno tras otro todos sus elementos hijos de forma horizontal o vertical según se establezca su propiedad *android:orientation*

asigna a cada uno. Los atributos de este estilo son los siguientes: altura 240 dp, ancho ajustado al contenido, tamaño del texto 20 dp, estilo del texto en negrita, margen (espacio alrededor del botón) de 5 dp y *padding*(espacio entre el texto y el borde del botón) de 12 dp.

Se establece habilitado *BotonConectar* y deshabilitado el resto (figura 3.1), pues ésta es la situación que se desea al iniciarse la aplicación.

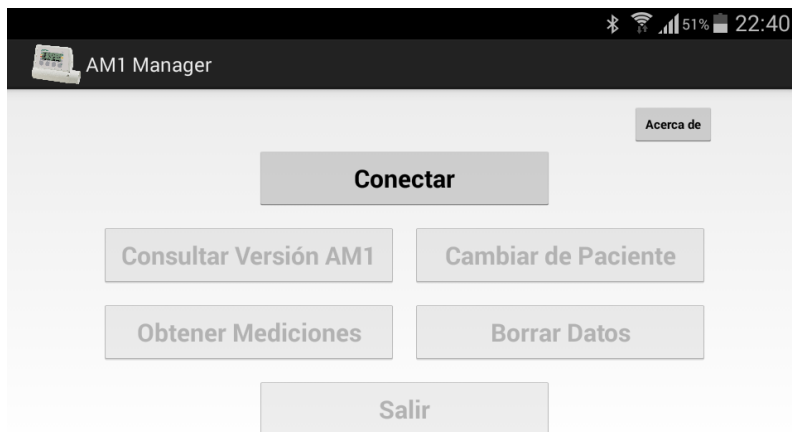


Figura 3.1: Pantalla de inicio de la aplicación.

### 3.2. *activity\_grafica.xml*

Ésta es la pantalla que se asocia a la clase *ActivityGrafica*.

Consta de un *LinearLayout*, que ocupa toda la pantalla y que contiene un único elemento: la gráfica que se muestra al usuario (figura 3.2). Se trata de un control de tipo *XYPlot*, al que se le asigna el identificador *Grafica*. Sus propiedades ancho y alto se establecen para que ocupe todo el espacio disponible.

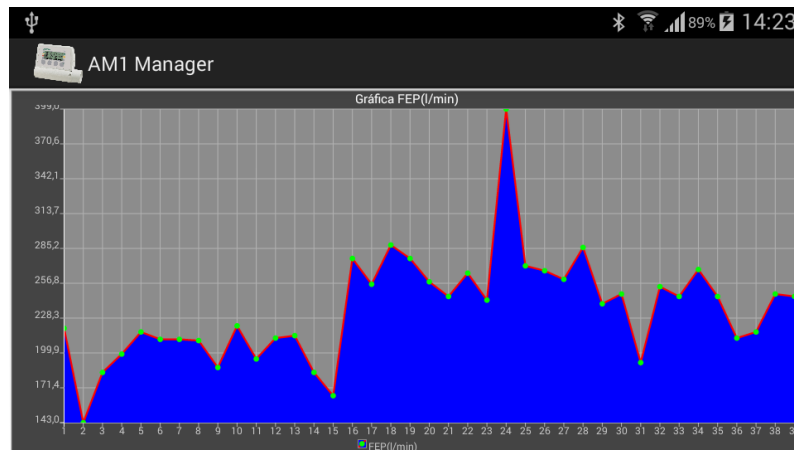


Figura 3.2: Gráfica con los valores del parámetro PEF.

### 3.3. *mostrar\_datos.xml*

Esta interfaz está diseñada como soporte para ofrecer al usuario los datos de las mediciones procedentes del espirómetro. Este *layout* se recupera en el método *MostrarDatos(String[] datos)* y se añade a un cuadro de diálogo, que finalmente se muestra.

El *layout* es de tipo *LinearLayout*, definido para ocupar todo el espacio disponible y con orientación vertical. Cuenta con los siguientes elementos:

- Una etiqueta (*TextView*) cuyo *id* es “Paciente”, altura ajustada al contenido, todo el espacio disponible de ancho y texto de tamaño 15 dp, centrado y estilo cursiva. En este *TextView* se muestra el nombre del paciente y el número de mediciones.
- Un *GridView*<sup>6</sup>, de nombre *GridEncabezado*, con nueve columnas, espacio horizontal entre celdas de 3 dp y vertical de 5 dp, alto ajustado al contenido y

<sup>6</sup>Un *GridView* es un control de tipo tabular, es decir, presenta elementos divididos en filas y columnas.

ancho el máximo posible. Además se indica que el espacio horizontal sobrante será absorbido a partes iguales por las columnas. Esta tabla, que tendrá una única fila, se usa para mostrar los nombre de los parámetros y sus unidades.

- Un control idéntico al anterior, pero con identificador *GridMediciones*, que se utiliza para para mostrar los valores de cada parámetro.

En la figura 3.3 se puede observar el *layout*, ya dentro del cuadro de diálogo y relleno con los datos.



Resultados Espirometría								
Nombre del Paciente: Carlos Contreras - Nº Mediciones: 105								
FVC (ml)	FEP (l/min)	FEV1 (ml)	FEF75 (ml/s)	FEF50 (ml/s)	FEF25 (ml/s)	FEF7525 (ml/s)	Fecha	Hora
3173	220	2917	2261	2771	3587	2749	03.09.14	15.34
2232	143	1710	1513	1360	1649	1460	03.09.14	15.34
2551	184	2551	2890	2941	3043	2890	03.09.14	15.34
1813	199	1813	2958	3077	3315	3084	03.09.14	15.34
2654	217	2645	3026	3383	3468	3317	04.09.14	17.55

Volver

Gráfica FEP

Figura 3.3: Resultados de la espirometría.

### 3.4. *textview\_datos.xml*

En este fichero se define el elemento que se usará en cada celda del GridView *GridMediciones*. Se trata de una etiqueta con texto centrado y tamaño de 15 dp. Además se indica que ocupa todo el ancho disponible y la altura se ajusta al texto.

### 3.5. *textview\_encabezado.xml*

En *textview\_encabezado.xml* se define la etiqueta que se replicará en las celdas de *GridEncabezado*. Sus propiedades son las mismas que la anterior, a las que se añade el estilo de texto negrita, como puede verse en la figura 3.3.



# Capítulo 4

## Lógica de la Aplicación

En este capítulo se describen cada una de las clases en las que se estructura la parte lógica del proyecto. En la figura 4.1 se puede observar un diagrama de clases del proyecto completo.

### 4.1. Clase *MainActivity*

En esta clase se desarrolla toda la lógica de la *Activity* (pantalla) principal. Desde aquí se accede a los elementos que componen la interfaz de la pantalla. La clase tiene los siguientes atributos:

- Un atributo de tipo *Button* que se usa para desactivar/activar el *BotonConectar*.
- Un atributo de tipo *BluetoothAdapter* para comprobar si el dispositivo tiene soporte para Bluetooth.
- Un atributo *BroadcastReceiver* que se utiliza para detectar si el usuario desactiva el Bluetooth.

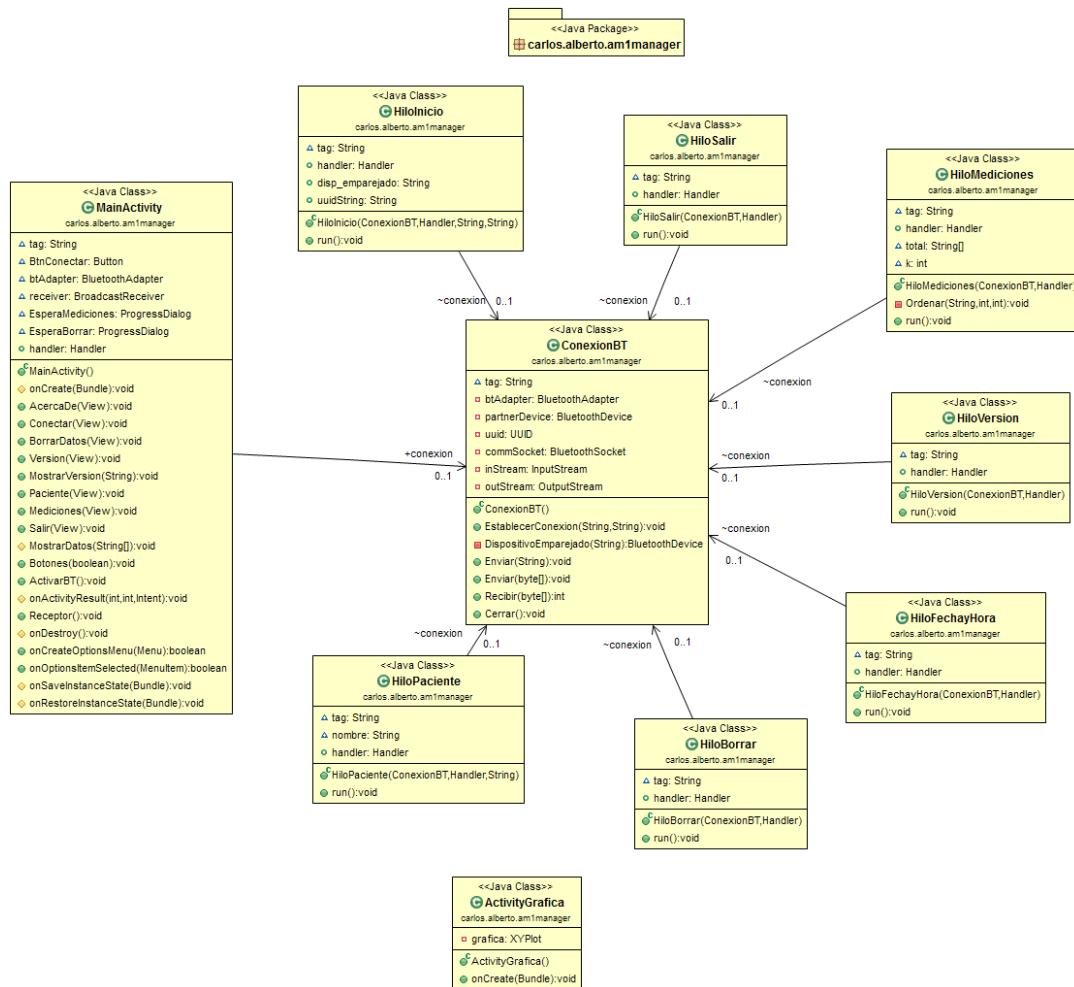
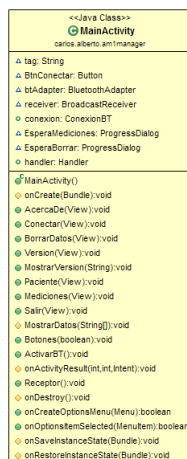


Figura 4.1: Diagrama de clases del proyecto.

- Dos *ProgressDialog* que se muestran a modo de espera mientras se obtienen las mediciones o se borran los datos respectivamente.
- Una instancia de la clase *ConexionBT*. Este atributo, al que se le da el nombre *conexion*, se pasa como parámetro a cada hebra.

En la figura 4.2 se puede observar un diagrama de esta clase, con los atributos descritos anteriormente y los métodos definidos, que se detallan a continuación.

Figura 4.2: Diagrama de la clase *MainActivity*.

#### 4.1.1. Método *onCreate(Bundle savedInstanceState)*

Este método se ejecuta nada más comenzar la aplicación.

En primer lugar se vincula la Activity con la interfaz *activity\_main.xml* y se guarda una referencia del *BotonConectar* mediante el método *findViewById(int id)* en el atributo creado anteriormente para ser usada en otros métodos de la clase.

A continuación se comprueba si el dispositivo es capaz de usar Bluetooth. En caso negativo se muestra un mensaje al usuario y se cierra la aplicación. Si no es así, se comprueba si Bluetooth está desactivado. En caso de darse tal circunstancia se llama al método *ActivarBT()*.

Por último se invoca el método *Receptor()*. Dicho método se describe en el apartado 4.1.15.

#### 4.1.2. *Handler*

Se define un *handler* para intercambiar datos en entre los hilos que se ejecutan en segundo plano y el hilo principal o *UI Thread*. En la clase *MainActivity* se implementa el método *handleMessage(Message msg)* que recibe como parámetro el

mensaje enviado desde un hilo. Se obtiene el *bundle* contenido en el mensaje y se extrae el valor de tipo *String* asociado a la clave “Operacion”. En función de dicho valor se ejecutan las operaciones que a continuación se detallan para cada caso.

- **Caso “Inicio”.** Se comprueba si se ha producido un error en la hebra que ha enviado el mensaje extrayendo el *boolean* asociado a la clave “Error”. Si ha habido error, se muestra un mensaje al usuario y se habilita el *BotonConectar*. En caso contrario, se notifica al usuario que se ha conectado con el espirómetro, se instancia una hebra de tipo *HiloFechayHora* y se lanza.
- **Caso “Versión”.** Si ha habido error, se muestra un mensaje al usuario y se habilita el *BotonConectar*. En caso contrario, se llama al método *MostrarVersion* pasándole como parámetro los datos que previamente se han obtenido del mensaje enviado por la hebra *HiloVersion*.
- **Caso “Borrar”.** En primer lugar se cancela el cuadro de espera iniciado al pulsar *BotonBorrar*. A continuación se muestra al usuario la cadena obtenida del mensaje enviado por la hebra *HiloBorrar* que informa si hubo éxito en la acción o no. Por último se extrae el valor asociado a la clave “Error”. Si se produjo error, se habilita *BotonConectar* y si no, se habilitan el resto de botones con *Botones(true)*.
- **Caso “MsjToast”.** Se muestra un mensaje *Toast*<sup>1</sup> con la cadena obtenida del mensaje. Si se produjo un error, se habilita *BotonConectar* y si se completó la operación con éxito, se habilitan el resto de botones.
- **Caso “Mediciones”.** En primer lugar se cancela el cuadro de espera iniciado al pulsar el *BotonMediciones*. A continuación se comprueba si se produjo un

---

<sup>1</sup>Un Toast es un mensaje que se muestra en pantalla durante unos segundos al usuario para luego volver a desaparecer automáticamente sin requerir ningún tipo de actuación por su parte

error extrayendo el valor *boolean* con clave “Error”. Si éste informa de un error, se muestra un mensaje informativo al usuario y se habilita el *BotonConectar*. En caso de éxito, se extrae el número de mediciones del mensaje. Si el número es superior a cero, se obtienen los datos y se llama al método *MostrarDatos* pasándole como parámetro los datos de las mediciones extraídas anteriormente. En caso de que no haya mediciones en la memoria del *AM1* se notifica al usuario y habilitan los botones.

- **Otro Caso.** Por seguridad se añade esta rama por defecto. Si el programa ejecuta este código se ha producido un error así que se informa de ello con un mensaje *Toast*. También se habilita el *BotonConectar*.

#### 4.1.3. Código del Atributo *onClick* de *BotonAcercaDe*

Esta parte de código se ejecutará cuando el usuario pulse el *BotonAcercaDe*.

Se instancia un objeto de la clase *AlertDialog*, se añade la cadena “Acerca de” como título, la información sobre la aplicación como mensaje y un botón para cancelar el cuadro de diálogo con el texto “Aceptar”. Por último se muestra. (Figura 4.3)

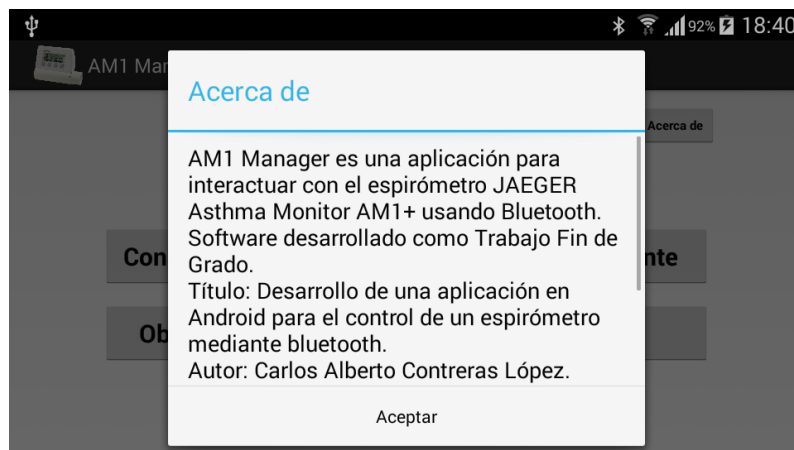


Figura 4.3: Información mostrada al pulsar *BotonAcercaDe*.

#### 4.1.4. Código del Atributo *onClick* de *BotonConectar*

Se declaran las variables de tipo *String nombre\_disp* con valor “Asthma Monitor” y *uuid* con valor “00001101-0000-1000-8000-00805F9B34FB”. A continuación se deshabilita el *BotonConectar*. Por último se instancia una hebra de la clase *HiloInicio* a la que se le pasan como parámetros el atributo de la clase *ConexionBT*, el *handler* y las dos variables locales mencionadas anteriormente y se lanza.

#### 4.1.5. Código del Atributo *onClick* de *BotonVersion*

Al pulsar el *BotonVersion* se desactivan los botones, se instancia una hebra de tipo *HiloVersion* y se lanza. En la figura 4.4 se puede observar a grandes rasgos la secuencia que sigue el programa al pulsar el *BotonVersion*.

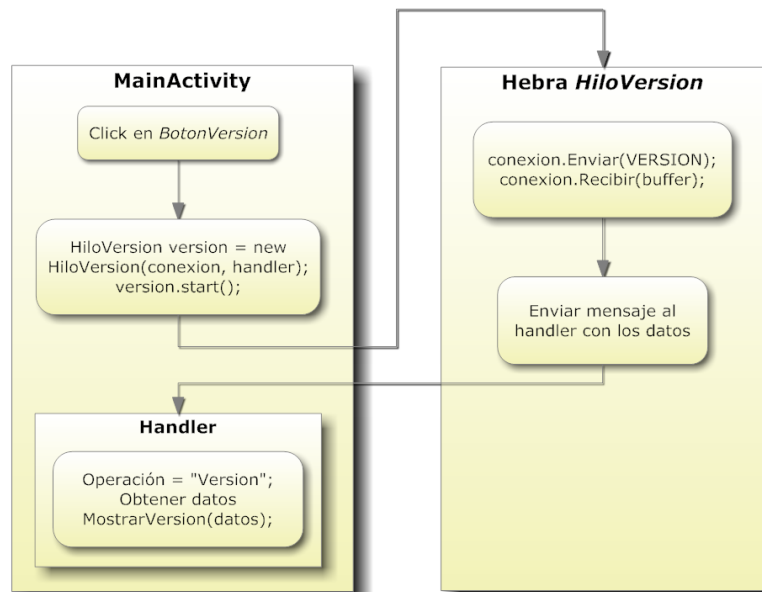


Figura 4.4: Secuencia cuando el usuario pulsa *BotonVersion*.

#### 4.1.6. Método *MostrarVersion(String cadena)*

Este método es llamado desde el *handler* pasándole como parámetros la cadena enviada por la hebra *HiloVersion*. Las operaciones que realiza son las siguientes:

Se instancia un objeto de la clase *AlertDialog*, se añade la cadena “Versión” como título, la versión del espirómetro recibida como parámetro a modo de mensaje y un botón con el texto “Aceptar” que al ser pulsado cancela el cuadro de diálogo y habilita los botones. Por último se muestra haciendo uso del método *show()*. El resultado se puede ver en la figura 4.5.

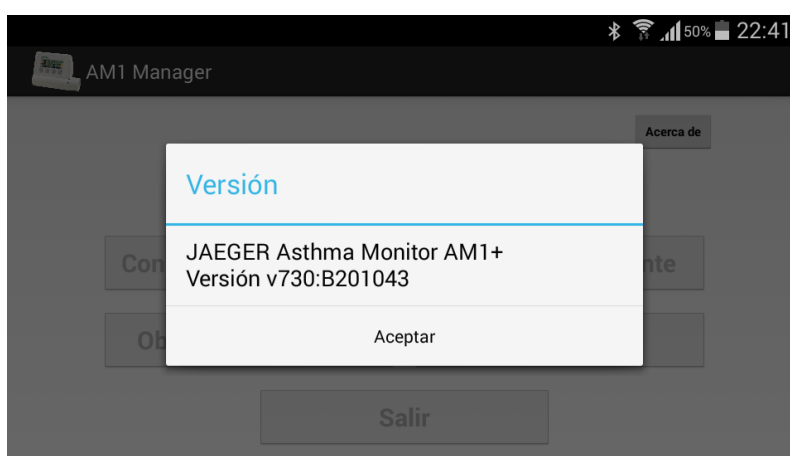


Figura 4.5: Información mostrada al pulsar *BotonVersion*.

#### 4.1.7. Código del Atributo *onClick* de *BotonPaciente*

En primer lugar se desactivan los botones y se crea un *EditText* (cuadro para escribir texto) para que el usuario pueda escribir el nuevo nombre del paciente. Este *EditText* se limita a un máximo de veinte caracteres pues la memoria del espirómetro no es capaz de almacenar más.

A continuación se crea un cuadro de diálogo (*AlertDialog*). A éste se añaden el título “Cambiar Paciente”, el mensaje “Introduzca el Nombre del Paciente:”, el

*EditText* declarado previamente y dos botones. El primero, con el texto “Aceptar” realiza las siguientes acciones al ser pulsado: se obtiene el texto introducido por el usuario, se instancia una hebra de la clase *HiloPaciente* a la que se le pasa como parámetros el *handler*, el atributo *conexion* de la clase *ConexionBT* y la cadena con el nombre del paciente introducido y se lanza dicha hebra. El segundo botón, con el texto “Cancelar”, se usa para cerrar el cuadro de diálogo y habilitar los botones con objeto de que el usuario pueda realizar nuevas acciones. Por último se muestra el *AlertDialog* haciendo uso del método *show()*.

#### 4.1.8. Código del Atributo *onClick* de *BotonMediciones*

Al pulsar sobre el botón *BotonMediciones* se deshabilitan los botones, se instancia una hebra de tipo *HiloMediciones*, se muestra un cuadro de espera con el texto “Obteniendo Mediciones...” y se lanza la hebra.

#### 4.1.9. Método *MostrarDatos(String[] datos)*

Este método se encarga de mostrar los datos de la espirometría, una vez obtenidos en el *handler* procedentes del mensaje enviado por la hebra *HiloMediciones*. Recibe como parámetro un array con elementos de tipo *String*. El primero de ellos es el nombre del paciente y el resto los valores de los parámetros de cada medición.

En primer lugar se recupera el *layout* definido en *XML* con los controles adecuados para mostrar los datos utilizando el método *inflater* y se almacena en una variable de tipo *View* llamada *MostrarMediciones*.

Se accede al *TextView* (etiqueta) definido en el *layout* y se escribe en él el nombre del paciente (primer elemento del array *datos*) seguido del número de mediciones, calculadas a partir del tamaño de *datos*.

A continuación, se elimina la primera posición del array y se define un array con elementos de tipo *String* con el encabezado de la tabla de datos (los nombres de



cada parámetro y las unidades en las que se miden).

Posteriormente se añaden los datos al *GridView* (control tabular) mediante los siguientes pasos:

- Se crea un adaptador *ArrayAdapter* con elementos de tipo *String* al que se le pasan los datos y el control que se representará en cada celda de la tabla, en este caso una etiqueta llamada *textview\_datos* definida en un archivo *XML*.

El código usado es el siguiente:

```
ArrayAdapter<String>adaptador = new ArrayAdapter<String>(this,  
R.layout.textview_datos, DatosSinNombre);
```

- Se obtiene una referencia al *GridView* y se le añaden los datos por filas mediante las siguientes instrucciones:

```
final GridView gridMediciones = (GridView)MostrarMediciones.  
findViewById(R.id.GridMediciones);  
gridMediciones.setAdapter(adaptador);
```

Para añadir el encabezado se procede de forma análoga.

Como último paso, el método se encarga de mostrar todos los datos en un cuadro de diálogo. Para ello se instancia un objeto de la clase *AlertDialog* al que se le añaden el título “Resultados Espirometría” y la variable de tipo *View* que contiene todos los datos que se han añadido anteriormente. También se definen dos botones en el cuadro de diálogo. Al primero se le añade el texto “Gráfica FEP”. Al pulsar sobre él, además de habilitar los botones de la aplicación, se lanza una nueva *Activity* (pantalla) para mostrar una gráfica con los datos del parámetro *FEP*. A esta nueva *Activity* se le pasa el array con los datos de todas las mediciones, excepto el nombre del paciente. El segundo botón, con el texto “Volver” cancela el *AlertDialog* y habilita los botones para que el usuario pueda realizar nuevas acciones. Finalmente se muestra

el *AlertDialog* de forma que ocupe toda la pantalla del dispositivo.

#### 4.1.10. Código del Atributo *onClick* de *BotonBorrar*

Cuando el usuario hace click en el *BotonBorrar* se realizan las acciones que se comentan a continuación.

En primer lugar se desactivan los botones. Seguidamente se instancia un objeto de la clase *AlertDialog* al que se le añaden el título “Borrar Datos”, el mensaje “¿Quiere borrar la memoria del AM1?” y dos botones. El primero, con el texto “Sí”, se usa para confirmar el borrado: se instancia una hebra *HiloBorrar*, se muestra un cuadro de espera con el mensaje “Borrando Datos...” y se lanza la hebra. Al segundo botón se le añade el texto “Volver”. Al ser pulsado se cierra el cuadro de diálogo y se habilitan los botones. Por último se llama al método *show()* para mostrar el *AlertDialog*. En la figura 4.6 se puede observar la ejecución del programa tras pulsar *BotonBorrar*.

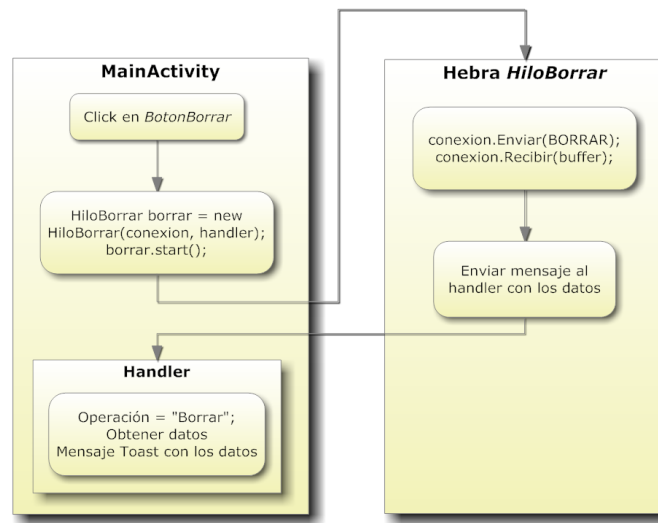


Figura 4.6: Secuencia cuando el usuario pulsa *BotonBorrar* y confirma.

#### 4.1.11. Código del Atributo *onClick* de *BotonSalir*

Para salir de la aplicación se pedirá confirmación al usuario. Al pulsar sobre el *BotonSalir* en primer lugar se deshabilitan los botones. A continuación se crea un cuadro de diálogo con el título “Salir”, el mensaje “¿Está seguro de que quiere salir?” y dos botones. Al de confirmación se le añade el texto “Sí”. Éste botón instancia una hebra de tipo *HiloSalir* y la lanza para apagar el espirómetro y cierra la aplicación invocando *finish()*. El segundo botón, con el texto “Cancelar” tiene como objeto cerrar el cuadro de diálogo y habilitar los botones. La última acción que se realiza es mostrar el *AlertDialog*. En la figura 4.7 se puede observar la ejecución del programa tras pulsar *BotonSalir*.

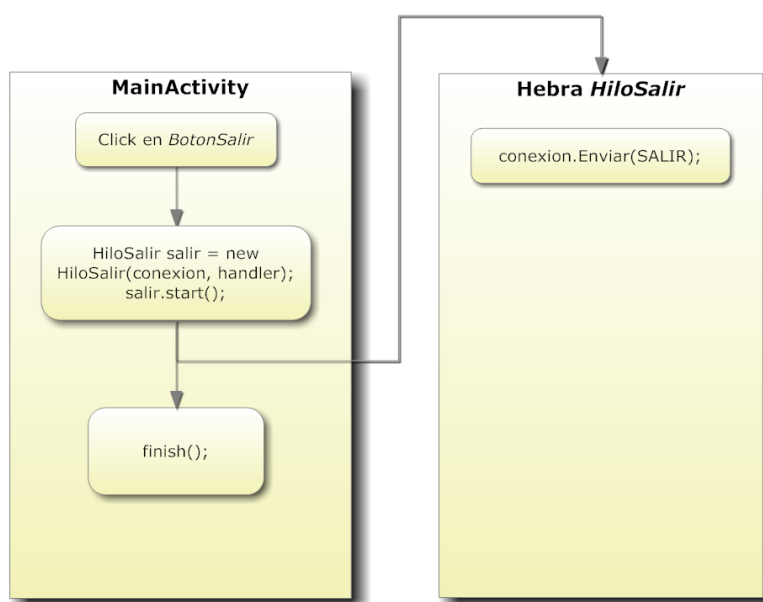


Figura 4.7: Secuencia cuando el usuario pulsa *BotonSalir* y confirma.

#### 4.1.12. Método *Botones(boolean estado)*

Este método se usa para habilitar/deshabilitar los botones *BotonVersion*, *BotonPaciente*, *BotonMediciones*, *BotonBorrar* y *BotonSalir*. Para ello se obtiene una referencia a cada uno de ellos con el método *findViewById(int id)* y se selecciona su estado con *setenable(estado)*.

#### 4.1.13. Método *ActivarBT()*

Con este método se muestra al usuario una petición para que active Bluetooth. (Figura 4.8). Se usa el siguiente código:

```
startActivityForResult(new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE),  
0);
```

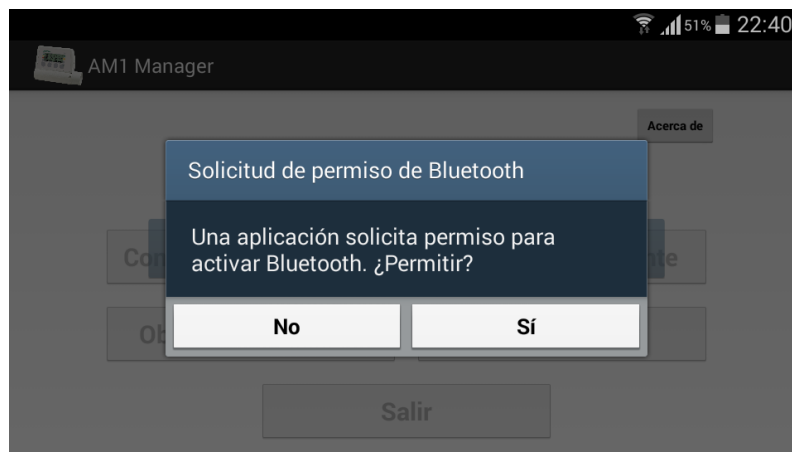


Figura 4.8: Petición de activación de Bluetooth.

#### 4.1.14. Método *onActivityResult(int requestCode, int resultCode, Intent data)*

Este método recibe la contestación del usuario a la petición de activación de Bluetooth. En caso de que haya respondido negativamente, se notifica que es necesario que Bluetooth esté activo y se cierra la aplicación.

#### 4.1.15. Método *Receptor()*

En este método se define un *BroadcastReceiver* destinado a “escuchar” cambios en el estado de Bluetooth en el dispositivo. En este caso se comprueba si se ha apagado y, si es así, se llama a *ActivarBT()*. De esta forma, si el usuario apaga Bluetooth, inmediatamente se le requiere que lo active para poder continuar.

Por último, es necesario registrar el receptor usando el método *registerReceiver*.

#### 4.1.16. Método *onDestroy()*

Puesto que anteriormente hemos registrado un *BroadcastReceiver*, en el método *onDestroy()* es necesario hacer la operación contraria, con *unregisterReceiver*.

#### 4.1.17. Método *onCreateOptionsMenu(Menu menu)*

En este método se añaden las opciones que aparecerán cuando se pulse el botón de opciones del dispositivo. Simplemente se añade una opción con el texto “Salir” y el identificador 1.

#### 4.1.18. Método *onOptionsItemSelected(MenuItem item)*

En este método se ejecutan las acciones una vez el usuario haya seleccionado una opción en el menú.

Se obtiene la opción seleccionada aplicando *getItemId()* al parámetro del método. Después se comprueba si es 1 (opción “Salir”) y en ese caso se finaliza la aplicación con *finish()*.

#### 4.1.19. Método *onSaveInstanceState(Bundle EstadoGuardado)*

Este método se usa para guardar el estado (habilitados o no) de los botones de la aplicación antes de que se apague la pantalla del dispositivo. Esto es necesario porque en Android, al apagarse la pantalla, la *Activity* es destruida y, al restaurarse la aplicación, se llama a *onCreate*, de forma que la vista que se obtiene es la misma que al comenzar la aplicación.

Para guardar el estado de los botones, basta con consultar el del *BotonConectar*, pues el estado del resto es siempre el contrario. Se obtiene el estado de dicho botón y se añade al *bundle EstadoGuardado*, parámetro de este método, mediante una variable de tipo *boolean*.

#### 4.1.20. Método *onRestoreInstanceState(Bundle EstadoGuardado)*

Este método se ejecuta al restaurarse la aplicación tras un periodo de reposo. Las acciones que se realizan son recuperar del *bundle* el estado del *BotonConectar*, establecer el estado de éste con el valor obtenido; y el del resto del botones con el opuesto.

## 4.2. Clase *ConexionBT*

Esta clase encapsula todo lo necesario para establecer una conexión Bluetooth, así como enviar y recibir datos a través de la conexión previamente abierta. En la clase *MainActivity* se instancia un objeto de esta clase y se pasa a cada una de las hebras, en las que se hará uso de los métodos *Enviar* y *Recibir* definidos en esta clase. En la figura 4.9 se muestra un diagrama de la clase *ConexionBT*.

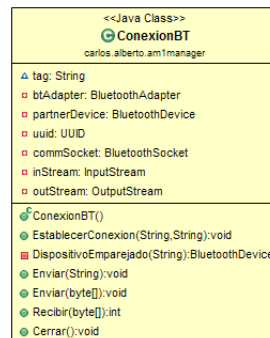


Figura 4.9: Diagrama de la clase *ConexionBT*.

La clase cuenta con los siguientes atributos:

- Un atributo de tipo *BluetoothAdapter* que representa el dispositivo local de Bluetooth, esto es, el dispositivo sobre el que se ejecuta la aplicación.
- Un atributo de tipo *BluetoothDevice* que representa el dispositivo con el que se conectará la aplicación, es decir, el espirómetro *AM1*.
- Un atributo de tipo *UUID*<sup>2</sup> que identifica el servicio Bluetooth. Este identificador lo define el servidor (el espirómetro) y el cliente (la aplicación) se conecta haciendo uso de él.

---

<sup>2</sup>El UUID (universally unique identifier) es un patrón de 128 bits que consta de cinco grupos de dígitos hexadecimales con 8, 4, 4, 4 y 12 dígitos, respectivamente.

- Un atributo de tipo *BluetoothSocket* que representa el *socket*<sup>3</sup> que se establece entre los dispositivos conectados.
- Un atributo de tipo *InputStream* que representa el flujo de entrada del *socket*.
- Un atributo de tipo *OutputStream* que representa el flujo de salida del *socket*.

En la clase se definen varios métodos, que se describen a continuación.

#### 4.2.1. Constructor *ConexionBT()*

En el constructor se obtiene el adaptador Bluetooth del dispositivo donde se ejecuta la aplicación y se guarda en el atributo de tipo *BluetoothAdapter*.

#### 4.2.2. Método *BluetoothDevice DispositivoEmparejado(String nombre\_disp)*

Este método compara el nombre pasado como parámetro (“Asthama Monitor”) con los nombres de los dispositivos emparejados. Si alguno contiene<sup>4</sup> dicho nombre se devuelve el dispositivo encontrado. Si no hay dispositivos emparejados o ninguno contiene el nombre se devuelve *null*.

---

<sup>3</sup>Un socket es una interfaz a una red de datos. Un programa puede leer datos entrantes de un socket y escribir datos de salida en el mismo sin necesidad de conocer el funcionamiento interno de la red.

<sup>4</sup>Sólo se pide que el dispositivo emparejado contenga “Asthama Monitor” para que la aplicación sea válida para todos los espirómetros del modelo ya que el nombre de cada unidad es “Asthama Monitor” seguido de su número de serie.



### 4.2.3. Método *EstablecerConexion(String nombre\_disp, String uuidString)*

Este método tiene dos parámetros de tipo *String*: el nombre del dispositivo al que hay que conectarse y el UUID del servicio Bluetooth. En su encabezamiento se añade *throws IOException* para indicar que las posibles excepciones que se produzcan se tratarán desde donde sea llamado el método dentro de un bloque *try-catch*.<sup>5</sup>

En primer lugar, se llama al método *DispositivoEmparejado(String nombre\_disp)* pasándole como parámetro el nombre del dispositivo y se guarda el valor devuelto por el método en el atributo de tipo *BluetoothDevice partnerDevice* declarado al inicio de la clase.

Posteriormente, se crea el *socket* aplicando a *partnerDevice* el método *createRfcommSocketToServiceRecord(UUID uuid)* al que se le pasa como parámetro el UUID recibido como parámetro y se guarda el *socket* en el atributo de tipo *BluetoothSocket*. A continuación se conecta con el espirómetro aplicando al *socket* el método *connect()*.

Por último se obtienen los flujos de datos de entrada y salida aplicando los métodos *getInputStream()* y *getOutputStream()* al *socket* y se asignan a los atributos *inStream* y *outStream*, de tipo *InputStream* y *OutputStream* respectivamente, declarados al inicio de la clase.

---

<sup>5</sup>Un bloque try-catch es un mecanismo de tratamiento de errores. Las instrucciones que pueden generar problemas se insertan dentro del bloque try. El programa ejecutará dichas instrucciones y, si se produce un error, saltará al bloque catch, donde se encuentran las instrucciones para tratar el error.

#### 4.2.4. Método *Enviar(byte[] datos)*

Este método se usa para enviar datos de tipo *array de bytes* aplicando el método de la clase *android.bluetooth write(byte[] datos)* al flujo de salida del *socket*.

El método usa la cláusula *throws IOException* de forma que los posibles errores se tratarán en el ámbito donde se invoque.

#### 4.2.5. Método *Enviar(String datos)*

Este método llama al del apartado 4.2.4 pasándole los *bytes* obtenidos de aplicar el método *getBytes()* al parámetro de tipo *String* datos.

Al igual que el método anterior hace uso de *throws IOException*.

#### 4.2.6. Método *int Recibir(byte[] buffer)*

Este método se utiliza para obtener datos del flujo de entrada del *socket* y guardarlos en el parámetro *buffer*. Además se devuelve el número de *bytes* recibidos. Todo ello se hace haciendo uso del método de la clase *android.bluetooth int read(byte[] buffer)*.

Como los métodos anteriores, este método usa *throws IOException*.

#### 4.2.7. Método *Cerrar()*

El método *Cerrar()* cierra el *socket* aplicándole el método *close()*. También hace uso de *throws IOException* para tratar las excepciones en el ámbito donde se invoque.

### 4.3. Hebra *HiloInicio*

Esta hebra se encarga de conectar con el espirómetro y enviar el resultado de su operación a la *ActivityMain*. En la figura 4.10 se puede observar un diagrama de la

clase *HiloInicio*.

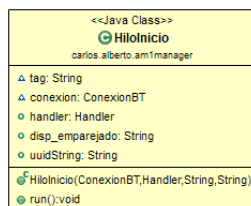


Figura 4.10: Diagrama de la clase *HiloInicio*.

Se definen un atributo de tipo *ConexionBT* llamado *conexion* para hacer las operaciones de enviar/recibir, otro de tipo *Handler* para enviar los datos al *handler* de la Actividad principal y dos de tipo *String* para el nombre del dispositivo y el *UUID* en formato cadena de texto.

#### 4.3.1. Constructor *HiloInicio(ConexionBT conexion, Handler handler, String disp\_emparejado, String uuidString)*

En el constructor de la clase se asignan los parámetros a los atributos de la clase.

#### 4.3.2. Método *run()*

El código de este método se ejecutará cuando se lance la hebra.

Se definen las variables locales *datos*, de tipo *String*, que se enviará a la *MainActivity*; *ENQ*, de tipo *array de bytes* e inicializada al valor 5; *buffer*, del mismo tipo, para almacenar los datos recibidos del *socket* Bluetooth y *error*, de tipo *boolean*, inicialmente a *false*, que se usa para informar a la *MainActivity* de si se produjo un error o no en esta hebra. La ejecución de dicha hebra puede verse en la figura 4.11.

Se usa un bloque *try-catch* para tratar las posibles excepciones. En el bloque *try* se aplica a *conexion* el método *EstablecerConexion(String nombre\_disp, String*

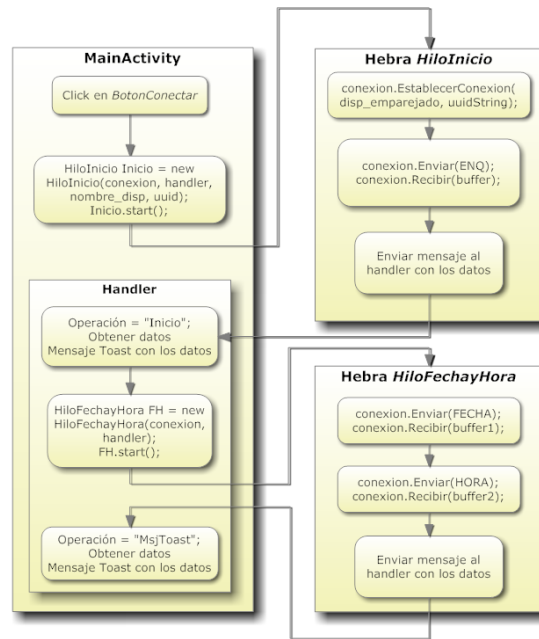


Figura 4.11: Secuencia cuando el usuario pulsa *BotonConectar*.

*uuidString*) con los parámetros nombre del dispositivo y *UUID* en texto, respectivamente. A continuación se envía *ENQ* para iniciar el intercambio de datos con el espirómetro. Éste contesta con un *ACK*<sup>6</sup> que se almacena en la variable *buffer* mediante el método *Recibir* de la clase *ConexionBT*. En el bloque *catch*, al que sólo se accede si se produjo un error, se pone a *true* la variable *error*. Seguidamente se comprueba que en *buffer* hay guardado efectivamente un *ACK* (valor 6). Si es así, se asigna a *datos* la cadena “¡Conectado!” y, en caso contrario, “No se pudo conectar con el AM1”. Por último, se crea un *bundle*<sup>7</sup> al que se le añaden la “Operación” “Inicio”, los datos y la variable *error*, se crea un nuevo mensaje, se añade el *bundle*

<sup>6</sup>Un *ACK* es un mensaje que el destino de la comunicación envía al origen de ésta para confirmar la recepción de un mensaje. En el caso del AM1 consiste en un entero de valor 6.

<sup>7</sup>Un *bundle* es un objeto que puede contener una lista de pares clave-valor con toda la información a pasar entre actividades o hebras.

al mensaje y se envía. Se usan las siguientes líneas de código:

```
Bundle bdl = new Bundle();
bdl.putString("Operacion","Inicio");
bdl.putBoolean("Error", error);
bdl.putString("Datos", datos);
Message msg = new Message();
msg.setData(bdl);
handler.sendMessage(msg);
```

## 4.4. Hebra *HiloFechayHora*

Esta hebra se encarga de establecer en el el espirómetro la hora y la fecha del dispositivo donde se ejecuta la aplicación y notificar a la *ActivityMain* el éxito o fracaso de la operación.

En la clase se definen dos atributos de tipo *ConexionBT* y *Handler*, respectivamente. Dichos atributos, además de los métodos definidos en esta clase, se reflejan en la figura 4.12.

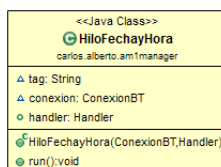


Figura 4.12: Diagrama de la clase *HiloFechayHora*.

### 4.4.1. Constructor *HiloFechayHora(ConstructorBT conexion, Handler handler)*

En el constructor de la clase se asignan los parámetros del mismo a los atributos.

### 4.4.2. Método *run()*

El método *run()* contiene las variables de tipo *array de bytes* *buffer1* y *buffer2* donde se almacenarán las confirmaciones tras hacer las operaciones de envío de fecha y hora, respectivamente; y una variable de tipo *String* llamada *datos* que se enviará a la actividad principal.

Los comandos que acepta el espirómetro para establecer la fecha y la hora son ‘D’ seguido de la fecha en formato “DD.MM.AA”<sup>8</sup> y ‘T’ seguido de la hora en formato “HH.MM”<sup>9</sup>, respectivamente. Además las cadenas deben terminarse con un retorno de carro.

Así pues, se obtiene la fecha del dispositivo instanciando un objeto de la clase *Date* y posteriormente se construyen las cadenas *FECHA* y *HORA*, previamente declaradas, con lo necesario para establecer fecha y hora en el espirómetro. Los formatos de fecha y hora requeridos se establecen con *SimpleDateFormat("dd.MM.yy")* y *SimpleDateFormat("HH.mm")* y después se aplican al objeto de la clase *Date*.

A continuación se envían los comandos *FECHA* y *HORA* y se reciben las confirmaciones, que quedan almacenadas en *buffer1* y *buffer2*. Seguidamente se comprueba que se han recibido correctamente los dos *ACK*. Si ambos son correctos se asigna a *datos* la cadena “Fecha y Hora sincronizadas” y en caso contrario se le asigna “No se pudo sincronizar Fecha y Hora”.

En último lugar se envía el mensaje hacia el *handler* que contiene un *bundle* al que se le han añadido previamente los pares clave-valor “Operación”-“MsjToast” y “Datos” junto a la variable *datos*.

<sup>8</sup>DD.MM.AA se refiere a la fecha usando dos dígitos para día, mes y año, respectivamente.

<sup>9</sup>HH.MM se refiere a la hora usando dos dígitos para horas (de 00 a 23) y minutos, respectivamente.

## 4.5. Hebra *HiloVersion*

Esta hebra se encarga de requerir al espirómetro el envío de su versión, obtenerla y enviarla a la *ActivityMain*.

La clase tiene los parámetros *conexion* y *handler*, de tipo *ConexionBT* y *Handler*, respectivamente y los métodos que se describen seguidamente. La figura 4.13 muestra un resumen de ello.



Figura 4.13: Diagrama de la clase *HiloVersion*.

### 4.5.1. Constructor *HiloVersion(ConexionBT conexion, Handler handler)*

En el constructor se asignan los parámetros a los atributos de la clase.

### 4.5.2. Método *run()*

El código de este método se ejecutará al lanzar una hebra de tipo *HiloVersion*.

Se definen las siguientes variables locales: *VERSION*, de tipo *String* e inicializada a 'v' seguido de un retorno de carro, que es el comando a enviar para solicitar al espirómetro el envío de su versión; *datos*, del mismo tipo, que es la cadena de texto que se enviará a la *MainActivity*; *buffer*, de tipo *array de bytes*, donde se almacena la información recibida del espirómetro; *NumBytes*, de tipo entero, donde se guarda el número de bytes recibidos; y *error*, de tipo boolean e inicialmente a *false*, que se usa para informar a la *MainActivity* de si se completó la acción con éxito o no.

Se usa un bloque *try-catch* para tratar una posible excepción. En el bloque *try* se envía el comando *VERSION*, se recibe la información del espirómetro y se guarda en *datos* con formato *String*. Para esta última acción se usa la siguiente línea:

```
datos = (new String(buffer)).substring(1,NumBytes-2);
```

Nótese que se eliminan el primer carácter (*ACK*) y los dos últimos (retorno de carro y salto de línea) haciendo uso de *substring*.

En el bloque *catch*, simplemente se asigna el valor *true* a la variable *error*.

Por último se envía el mensaje a la actividad principal. El mensaje contiene un *bundle* al que se le han añadido las variables *datos* y *error* y el par “Operacion”-“Version”, necesario para la correcta decodificación en el *handler*.

## 4.6. Hebra *HiloPaciente*

El cometido de esta hebra es establecer en la memoria del espirómetro el nombre del paciente introducido por el usuario y que se pasa como parámetro al instanciar dicha hebra. También se encarga de enviar el resultado de la operación a la *ActivityMain*. La figura 4.14 muestra un diagrama de esta clase.

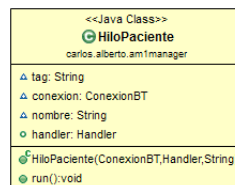


Figura 4.14: Diagrama de la clase *HiloPaciente*.

*HiloPaciente* tiene tres atributos: *conexion*, de tipo *ConexionBT*, necesario para hacer las operaciones de envío y recepción; *handler*, de tipo *Handler*, para enviar el mensaje a la actividad principal; y *nombre*, de tipo *String*, donde se almacena el nombre del paciente recibido como parámetro.



#### 4.6.1. Constructor *HiloPaciente(ConexionBT conexion, Handler handler, String nombre)*

El constructor de la clase asigna cada parámetro al atributo de la clase con el mismo nombre y tipo.

#### 4.6.2. Método *run()*

El método *run()* tiene definidas la variables *buffer*, de tipo *array de bytes*, para almacenar la confirmación enviada por el espirómetro; *datos*, de tipo *String*, que es la cadena que se envía a la *MainActivity*; y *error*, de tipo *boolean* para notificar del éxito o fracaso de la operación.

Lo primero que se hace es comprobar que todos los caracteres del nombre sean caracteres “Regular ASCII”, es decir, con valores entre 0 y 127. Si alguno no lo cumple se cambia por el carácter ‘?’. Se hace esto porque se ha comprobado que los caracteres con valor mayor a 127 producen fallos que conllevan a que no se visualicen correctamente los datos de la espirometría.

A continuación, se construye el comando necesario para cambiar el nombre del paciente en la memoria del *AM1*, consistente en el carácter ‘I’ seguido del nombre y terminado en un retorno de carro. Esta cadena se guarda en una variable de tipo *String* llamada *PACIENTE*.

Posteriormente, dentro de un bloque *try*, se envía el comando anterior y se recibe el *ACK* proveniente del espirómetro, el cual queda almacenado en *buffer*. En la parte del bloque *catch* se asigna el valor *true* a la variable *error*.

Antes de enviar el mensaje, se comprueba si *buffer* contiene el *ACK* (entero con valor 6). En caso positivo, se asigna a *datos* la cadena “Nombre del paciente cambiado con éxito” y, en caso negativo, se le asigna “No se pudo cambiar el nombre del paciente”.

Por último se añaden a un *bundle* las variables *datos* y *error*, y el par “Operacion”-“MsgToast” y se envía un mensaje a la *MainActivity* que contiene dicho *bundle*. En la figura 4.15 se muestra el cuadro de texto donde el usuario introduce el nuevo nombre del paciente que envía al *AM1*.

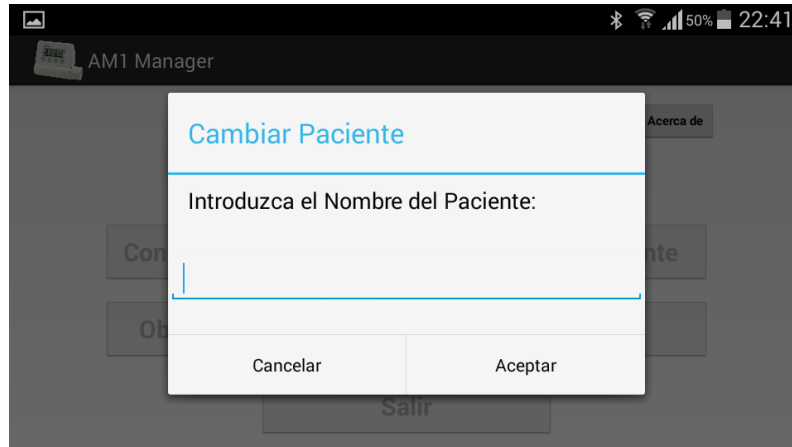


Figura 4.15: Cuadro de diálogo mostrado al pulsar *BotonPaciente*.

## 4.7. Hebra *HiloMediciones*

Esta hebra se encarga de de requerir los datos de las mediciones al espirómetro y enviarlos a la *ActivityMain*. En la figura 4.16 se puede observar un resumen de la clase.

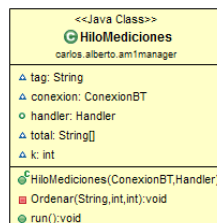


Figura 4.16: Diagrama de la clase *HiloMediciones*.

La clase tiene los atributos *conexión*, de tipo *ConexionBT*; *handler*, de tipo *Handler*; *total*, que es un array de elementos de tipo *String* donde se guardarán los datos ya ordenados; y *k*, de tipo entero, que se usa como índice para recorrer el array *total*.

#### 4.7.1. Constructor *HiloMediciones(ConexionBT conexion, Handler handler)*

En el constructor de la clase únicamente se asignan los parámetros del mismo a los atributos.

#### 4.7.2. Método *Ordenar(String buffer, int NumMed, int bloque)*

El espirómetro guarda los datos de las mediciones por parámetro; es decir, primero todos los datos del primer parámetro, después todos los del segundo, y así sucesivamente. Este método tiene como objetivo ordenar los datos por mediciones; esto es, primero los parámetros de la primera medida, después todos los de la segunda medida, etc. Los datos, ya ordenados de esta forma, quedan almacenados en el atributo *total*.

El método recibe los parámetros *buffer*, de tipo *String*, con los datos de un bloque<sup>10</sup> de medidas; *NumMed*, de tipo entero, que representa el número de medidas que hay en *buffer*; y *bloque*, también de tipo entero, que representa el número de bloque.

Se definen las siguientes variables locales: una de tipo *String* llamada *nombre* para el nombre del paciente; nueve arrays de elementos de tipo *String* para almacenar

---

<sup>10</sup>El espirómetro almacena hasta cuatro bloques, con un máximo de cien medidas cada uno. Los bloques deben ser requeridos uno a uno.

los datos de los siete parámetros, la fecha y la hora de cada medida; y dos enteros, llamados *n* e *i*, a modo de índices de los bucles que recorren los nueve arrays y el parámetro *buffer*, respectivamente.

Las acciones que se realizan en este método son las siguientes:

- Se recorre *buffer* carácter a carácter hasta encontrar un retorno de carro y se guarda el nombre del paciente en *nombre*.
- Se avanza por *buffer* y se extrae cada valor, guardándose en una posición del array correspondiente, según el parámetro del que se trate.
- Si se está ordenando el primer bloque, se copia *nombre* en la primera posición de *total*.
- Se copian ordenadamente todos los elementos de los nueve arrays en *total*, de forma que quedan ordenados por mediciones, es decir, primero los nueve parámetros de la primera medida, después los nueve de la segunda, etc.

Durante este algoritmo se aplica el método *String substring(int inicio, int fin)* a *buffer* para obtener secciones del mismo. En la secuencia de la figura 4.17 se pueden observar las llamadas a este método durante la ejecución de *HiloMediciones*.

### 4.7.3. Método *run()*

En este método se definen las siguientes variables:

- *STATUS*, de tipo *String* e inicializada a ‘C’ seguida de un retorno de carro, es el parámetro a enviar necesario para obtener el número de mediciones que hay en la memoria del *AM1*.
- *BLOQUE*, de tipo *array de bytes*, inicializado usando hexadecimales a  $\{0x42, 0x30, 0x31, 0x0d\}$ <sup>11</sup>, necesario para requerir el primer bloque de datos.

---

<sup>11</sup>Esta secuencia corresponde en ASCII a “B01” terminado en retorno de carro.

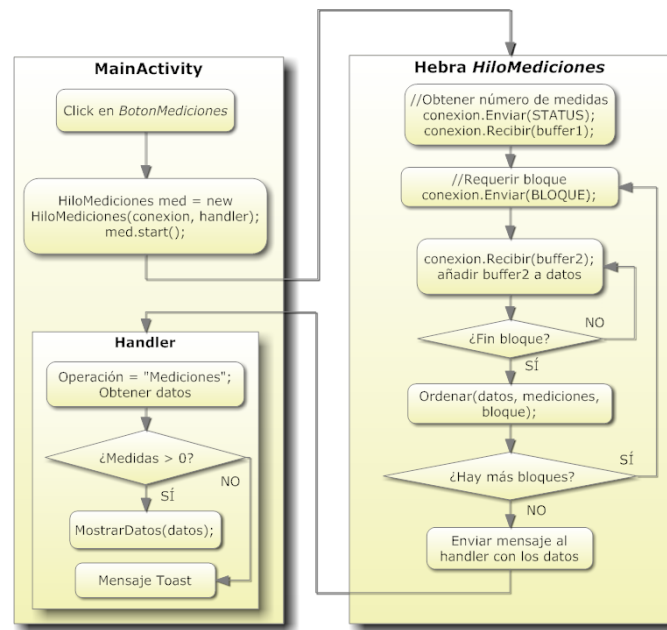


Figura 4.17: Secuencia cuando el usuario pulsa *BotonMediciones*.

- *buffer1* y *buffer2*, también de tipo *array de bytes*, para almacenar la información procedente del espirómetro relativa al número de mediciones y valores de mediciones de un bloque, respectivamente.
- Las variables de tipo entero *NumBloques*, para guardar el número total de bloques; *NumMediciones*, inicializada a cero, que almacena el número total de mediciones; y *mediciones*, que representa el número de mediciones que contiene un bloque concreto.
- Las variables de tipo *String auxiliar*, que se usa para guardar los datos obtenidos en una llamada al método *Recibir* de la clase *ConexionBT*, ya convertidos a este tipo; *datos*, donde se almacenan todos los datos de un bloque; y *FIN*, con valor “0098540” terminado en un retorno de carro y un salto de línea, que

es transmitido por el *AM1* al final de cada bloque de datos.

- Las variables de tipo *boolean error*, inicialmente a *false*, que se envía a la *MainActivity* para informar de la ocurrencia o no de un error; y *salir*, que se usa como condición de salida de un bucle.

Se usa un bloque *try-catch* para tratar la aparición de un posible error. En la parte *catch*, simplemente se asigna el valor *true* a la variable *error*. En el bloque *try* se realizan todas las acciones que a continuación se describen.

- Se envía el comando *STATUS* y se recibe el número de medidas que tiene almacenadas el *AM1*<sup>12</sup>. A continuación, con los datos obtenidos, se almacena en *NumMediciones*, ya en formato entero<sup>13</sup>, el número total de mediciones. Posteriormente, se calcula el número total de bloques teniendo en cuenta que, como máximo, un bloque contiene 100 medidas y se guarda este valor en la variable *NumBloques*.
- Se inicializa el array *total* con tamaño *NumMediciones* multiplicado por nueve, más uno.<sup>14</sup>

Si el número de mediciones es superior a cero, se ejecutan las acciones siguientes, que se repiten tantas veces como número de bloques haya.

- Se envía el comando *BLOQUE*, se incrementa en uno la tercera posición de la variable *BLOQUE*, para pedir el siguiente bloque en la próxima iteración en caso de que sea necesario y se asigna la cadena vacía a la variable *datos*.
- Se realizan tantas llamadas al método *Recibir* de la clase *ConexionBT* como sean necesarias para obtener un bloque completo. En cada operación *Recibir*

---

<sup>12</sup>El espirómetro almacena los datos usando el número correspondiente al carácter ASCII, aunque los datos se reciban en un buffer de tipo byte.

<sup>13</sup>Para convertir de ASCII a entero se usa el método *int getNumericValue(char c)*

<sup>14</sup>Nueve parámetros por cada medición, más el nombre del paciente.

se almacenan los bytes recibidos en *auxiliar*, convertidos a formato cadena de texto, y se añaden a *datos*, donde finalmente quedan los valores de todo un bloque de medidas.

- Se calcula el número de medidas que hay en *datos* y se guarda en la variable *mediciones*. Por último, se invoca el método *Ordenar*, al que se le pasan como parámetros *datos*, *mediciones* y *bloque*, que representa el número de bloque.

## 4.8. Hebra *HiloBorrar*

Esta hebra se usa para borrar la memoria del espirómetro. Además se encarga de notificar a la *ActivityMain* si hubo éxito o no en el borrado. En la figura 4.18 se muestra un resumen de esta clase.



Figura 4.18: Diagrama de la clase *HiloBorrar*.

La clase tiene los parámetros *conexión* y *handler*, de tipo *ConexionBT* y *Handler*, respectivamente.

### 4.8.1. Constructor *HiloBorrar(ConexionBT conexion, Handler handler)*

El constructor se encarga únicamente de asignar los parámetros a los atributos de la clase.

### 4.8.2. Método *run()*

El método *run()* tiene las variables *buffer*, de tipo *array de bytes*, donde se guarda la confirmación recibida del espirómetro; *datos*, de tipo *String*, que se enviará a la actividad principal; y *error*, que se usa para informar a la *MainActivity* de la aparición o no de un error durante la ejecución de esta hebra. Además también se define la variable de tipo *String BORRAR* que se inicializa a ‘L’ seguido de un retorno de carro. Éste es el comando que se debe enviar al *AM1* para borrar el contenido de su memoria.

Como en las demás hebras, se utiliza un bloque *try-catch*. En la parte del *try* se envía *BORRAR* y se recibe la confirmación, que queda almacenada en *buffer*. En la parte del *catch* se asigna el valor *true* a la variable *error*.

A continuación se comprueba si *buffer* contiene el *ACK*. Si es así, se asigna la cadena “Memoria del AM1 borrada” a la variable *datos*. En caso contrario, se le asigna la cadena “No se pudo borrar la memoria del AM1”.

Finalmente se añaden a un *bundle* los pares clave-valor “Operacion”-“Borrar”, “Datos”-variable *datos* y “Error”-variable *error* y se envía un mensaje a la *MainActivity* que contiene este *bundle*.

## 4.9. Hebra *HiloSalir*

Esta hebra se encarga de enviar al espirómetro el comando necesario para apagarlo. No se manda ningún mensaje a la actividad principal ya que la siguiente acción que se realiza es cerrar la aplicación.

Los atributos que se definen en esta clase son *conexión* y *handler*, de tipo *ConexionBT* y *Handler*, respectivamente. Estos atributos junto con los métodos de esta clase, detallados a continuación, están reflejados en la figura 4.19.



Figura 4.19: Diagrama de la clase *HiloSalir*.

#### 4.9.1. Constructor *HiloSalir(ConexionBT conexion, Handler handler)*

El constructor se encarga de asignar los parámetros a los atributos de la clase.

#### 4.9.2. Método *run()*

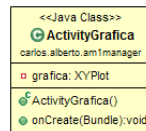
En el método *run()* se define la variable *SALIR*, de tipo *String* que se inicializa con el carácter ‘k’ seguido de un retorno de carro. Éste es el comando que hay que enviar al espirómetro para apagarlo.

A continuación se envía el comando *SALIR* dentro de un bloque *try*. La parte *catch* queda vacía. Por lo tanto se captura una posible excepción, aunque no se hace ninguna acción en caso de que se produzca.

### 4.10. Clase *ActivityGrafica*

La clase *ActivityGrafica* se encarga de mostrar una gráfica con los datos del parámetro *PEF*. Para ello se hace uso de la librería *androidplot*, la cual se ha tenido que añadir al proyecto pues no se encuentra disponible por defecto.

Se define un único atributo de tipo *XYPlot* llamado *grafica*. En la figura 4.20 se observa un diagrama de esta clase.

Figura 4.20: Diagrama de la clase *ActivityGrafica*.

#### 4.10.1. Método *onCreate(Bundle savedInstanceState)*

Este método se ejecutará nada más iniciarse esta *Activity*. En primer lugar se establece el *layout activity\_grafica.xml* definido previamente como pantalla de esta *Activity* y se obtiene una referencia al único elemento del *layout* (de tipo *XYPlot*) que se guarda en el atributo de la clase, con objeto de hacer uso de él posteriormente.

A continuación, se obtienen los datos procedentes de la *ActivityMain* (un array de elementos de tipo *String* con los valores de cada parámetro para cada una de las medidas) y se guardan en una variable llamada *datos* mediante las siguientes líneas:

```
Bundle bundle = getIntent().getExtras();
String datos[] = bundle.getStringArray("Datos");
```

Posteriormente se declaran las variables *EjeX* y *EjeY* como arrays de elementos de tipo *Number*<sup>15</sup> con tamaño igual al tamaño de *datos* dividido por nueve.<sup>16</sup> El paso siguiente es obtener los valores del parámetro *PEF* y almacenarlos en *EjeY*. Para hacer el cambio de *String* a entero se usa el método *int parseInt(String string)*. Los elementos de la variable *EjeX* se rellenan con números naturales consecutivos desde el uno hasta el número total de mediciones.

Finalmente, se añaden al atributo *grafica* las variables *EjeX* como valores del eje de abscisas y *EjeY* como valores del eje de ordenadas. Además se establece como título de la gráfica “Gráfica FEP(1/min)”. En la figura 4.21 se muestra la gráfica.

<sup>15</sup>Las variables *EjeX* y *EjeY* se declaran de tipo *Number* ya que la función para añadir datos a la gráfica así lo requiere. En una variable de este tipo se pueden guardar valores enteros.

<sup>16</sup>Se usa este tamaño ya que cada medida tiene siete parámetros además de la fecha y la hora.

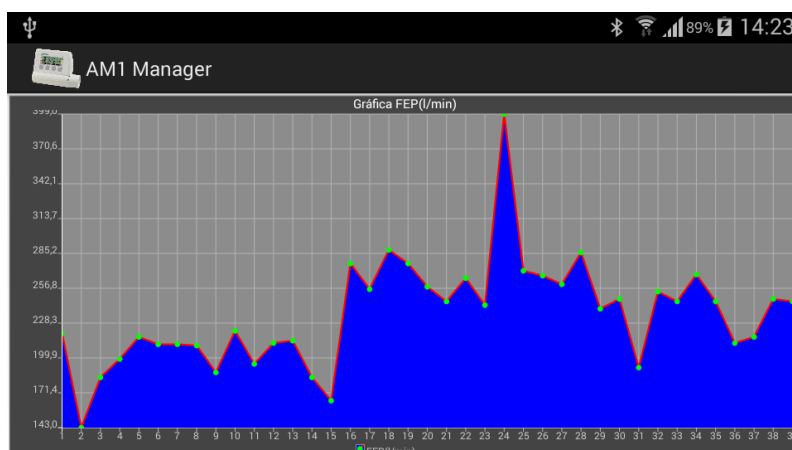


Figura 4.21: Gráfica con los valores del parámetro PEF.

## 4.11. Fichero *AndroidManifest.xml*

*AndroidManifest* es un fichero especial utilizado por toda aplicación *Android* que contiene la definición en XML de los aspectos principales de la aplicación, como por ejemplo su identificación (nombre, versión, icono...), sus componentes (pantallas, mensajes, etc), o los permisos necesarios para su ejecución.

Ha sido necesario declarar las dos *Activities*, *MainActivity* y *ActivityGrafica*, añadir el icono de la aplicación previamente guardado en el directorio *drawable*, indicar que las actividades funcionarán con la pantalla horizontal y explicitar que la aplicación puede acceder al Bluetooth del dispositivo donde se ejecute. Estos permisos se han añadido con las siguientes líneas:

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



# Capítulo 5

## Plan de Pruebas

En este capítulo se describen las pruebas realizadas para comprobar el correcto funcionamiento de la aplicación desarrollada. A continuación se enumeran todas las acciones de las que se ha obtenido constancia de su ejecución satisfactoria.

- Al iniciar la aplicación sin Bluetooth activado, ésta muestra el requerimiento de activación y responde correctamente a la respuesta del usuario.
- Si durante la ejecución de la aplicación es usuario deshabilita el Bluetooth, se muestra el requerimiento de activación.
- La pantalla inicial muestra todos los botones deshabilitados, salvo el *BotonConectar*.
- Tras conectar con el espirómetro, se deshabilita el *BotonConectar* y se habilitan el resto de botones. En caso de no poder conectar, no se realizan estas acciones y se muestra un mensaje al usuario notificando el error.
- El *BotonAcercaDe* muestra correctamente la información.
- El *BotonVersion* muestra correctamente los datos obtenidos del espirómetro. En caso de no poder obtener dichos datos, muestra el mensaje que informa de

que no se pudo obtener la versión del *AM1*.

- Al pulsar el *BotonPaciente* se muestra correctamente el cuadro de diálogo y el número de caracteres a introducir está limitado a un máximo de veinte. Cuando hay éxito en la operación, el nombre queda guardado en la memoria del *AM1* y, en caso de error, se informa al usuario.
- Al pulsar el *BotonMediciones* se lanza el cuadro de espera y se cierra antes de mostrar los datos. Los datos se muestran tal como están guardados en la memoria del espirómetro. En caso de no haber mediciones en la memoria, se avisa de esta situación al usuario. El número de mediciones se actualiza si se almacenan nuevas medidas.
- La gráfica del parámetro PEF muestra los datos correctamente. Se puede volver a la pantalla principal pulsando el botón atrás del dispositivo.
- Al pulsar el *BotonBorrar* se limpia efectivamente la memoria del *AM1*. En caso de no poder hacerlo se muestra la notificación adecuada.
- El *BotonSalir* cierra correctamente la aplicación, además de apagar el espirómetro.
- Los cuadros de diálogo sólo se pueden cancelar al pulsar sobre los botones habilitados para ello del propio diálogo y no con el botón atrás del dispositivo.
- Al pulsar sobre el botón opciones del dispositivo se despliega la opción salir, que cierra correctamente la aplicación.
- Tras un periodo de inactividad de la pantalla del dispositivo, el estado de los botones (habilitados o no) se conserva.
- Si no se consigue realizar alguna acción tras pulsar un botón, los botones vuelven al estado inicial (todos deshabilitados, excepto el *BotonConectar*).

En la tabla 5.1, se ofrecen los resultados de las pruebas realizadas con distintos dispositivos.

Dispositivo	Versión Android	Resultados
Samsung Galaxy mini	2.3.5	Funcionamiento correcto
Samsung Galaxy Ace S5830i	2.3.6	No es posible emparejarlo
Samsung Galaxy S4 mini	4.2.2	Se corta la conexión
Samsung Galaxy S4 mini	4.4.2	Se corta la conexión
Samsung Galaxy Young	4.1.1	Funcionamiento correcto
Motorola Moto G	4.4.4	Funcionamiento correcto
Nexus 4	4.4.4	Funcionamiento correcto
Tablet Archos 101G9	4.0.4	Funcionamiento correcto

Tabla 5.1: Resultados de las pruebas con varios dispositivos.

Con el dispositivo Samsung Galaxy Ace s5830i no es posible conectarse al espirómetro ya que carece del perfil bluetooth HID.

Con el terminal Samsung Galaxy S4 Mini, aunque se llega a conectar con el espirómetro, se corta la conexión Bluetooth a los pocos segundos. Esto ocurre tanto con la versión de Android 4.2.2 como con la versión 4.4.2.





# Conclusiones y Líneas Futuras

Tras la finalización del Trabajo Fin de Grado se puede afirmar que se han cumplido los objetivos inicialmente fijados. Se ha desarrollado una aplicación que puede ser ejecutada en dispositivos con sistema operativo Android. Dicha aplicación permite conectarse al espirómetro JAEGER Asthma Monitor AM1+ por medio de Bluetooth y realizar las operaciones de consulta de la versión del AM1, cambiar el nombre del paciente almacenado en el espirómetro, visualizar todos los parámetros de cada medición almacenada, obtener un gráfica de uno de los parámetros, borrar la memoria del AM1 y apagar el espirómetro cuando se sale de la aplicación. Además también se ha implementado una sincronización de fecha y hora entre el dispositivo Android y el AM1 que se ejecuta justo después de la conexión exitosa de los terminales.

Se ha construido, por un lado, los interfaces de usuario en XML y, por otro, se ha programado toda la lógica de la aplicación en Android de forma nativa (usando Java).

La metodología que se ha seguido ha sido la de resolver los problemas uno a uno y de menor a mayor complejidad. En primer lugar la aplicación se ha limitado a conectar ambos dispositivos y posteriormente se ha ido añadiendo funcionalidad (mediante la creación de nuevos botones en la interfaz y definiendo nuevas hebras). Por último se han añadido detalles que hacen la aplicación más vistosa de cara al usuario, como los cuadros de espera mientras se realiza una acción.

La realización de este trabajo ha permitido descubrir la programación en XML y

en un lenguaje tan popular actualmente como Java para Android. Además se han recordado conceptos de programación concurrente estudiados en una asignatura de la carrera. Por último se ha de resaltar también los conocimientos de L<sup>A</sup>T<sub>E</sub>X adquiridos durante la redacción del presente documento.

El espirómetro realiza unas funciones concretas y casi todas se han usado en esta aplicación. Es por ello que las ampliaciones de este Trabajo Fin de Grado son bastantes reducidas; aún así, a continuación se enumeran algunas de las posibles líneas futuras de trabajo:

- Añadir un botón para usar la función alarma del espirómetro.
- Generar una base de datos que se guardará en la memoria del dispositivo Android que contenga datos de medidas y nombres de pacientes.
- Generar un fichero de texto con las medidas de un paciente para poder compartirlo fácilmente.
- Extender la aplicación a otros sistemas operativos, como iOS<sup>1</sup> (iPhone Operating System) o Windows Phone, de manera que todos los usuarios de *smartphones* tengan acceso a la aplicación.
- Dar soporte en otros idiomas a la aplicación. Para cambiar de idioma se podría añadir la opción al botón opciones o crear un nuevo botón.
- Incorporar dentro de la aplicación la posibilidad de enviar un correo electrónico que contenga los valores de las mediciones.

---

<sup>1</sup>iOs es un sistema operativo móvil de la empresa Apple Inc originalmente desarrollado para el iPhone (iPhone OS), aunque después se ha usado en dispositivos como iPod Touch, iPad y Apple TV

# Apéndice A

## Manual de Usuario

En este apéndice se explica el funcionamiento de la aplicación desde el punto de vista del usuario final. En él se describen cada una de las opciones que presenta la aplicación al usuario.

**Emparejar el espirómetro con el dispositivo Android.** Antes de conectar es necesario emparejar el espirómetro con el dispositivo Android. Para ello, habilite el Bluetooth en el espirómetro (figura A.1) y haga una búsqueda del AM1. A continuación añada el AM1 a los dispositivos emparejados. La clave necesaria es 1609.

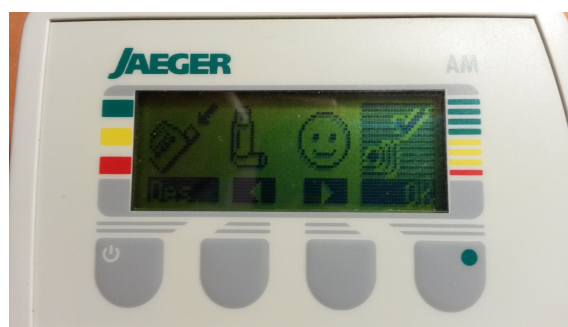


Figura A.1: Bluetooth del espirómetro activado.

**Conectar.** Una vez emparejado el espirómetro y con el Bluetooth del AM1 activado, haga click en “Conectar”. Si los dispositivos se conectan, se habilitarán el resto de botones, como puede observarse en la figura A.2.

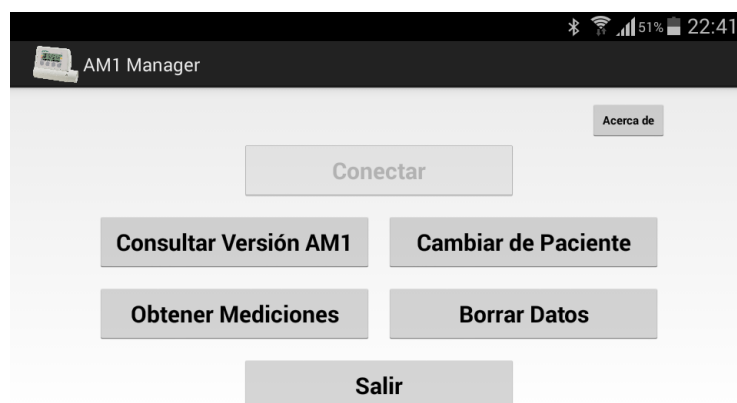


Figura A.2: Botones habilitados.

**Consultar Versión.** Para obtener la versión del espirómetro pulse “Consultar Versión AM1” y obtendrá una respuesta como la de la figura A.3.

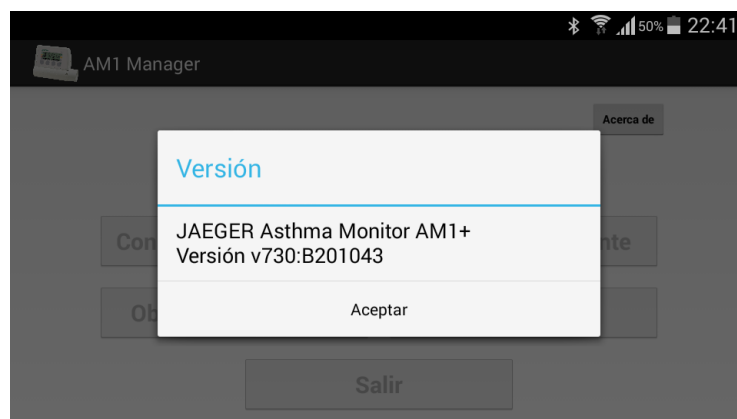


Figura A.3: Información acerca de la versión del espirómetro.

**Cambiar Nombre del Paciente.** Para cambiar el nombre del paciente haga click en “Cambiar Paciente”. Se abrirá un cuadro de diálogo como el de la figura A.4.

Introduzca el nombre (máximo veinte caracteres) y pulse aceptar.

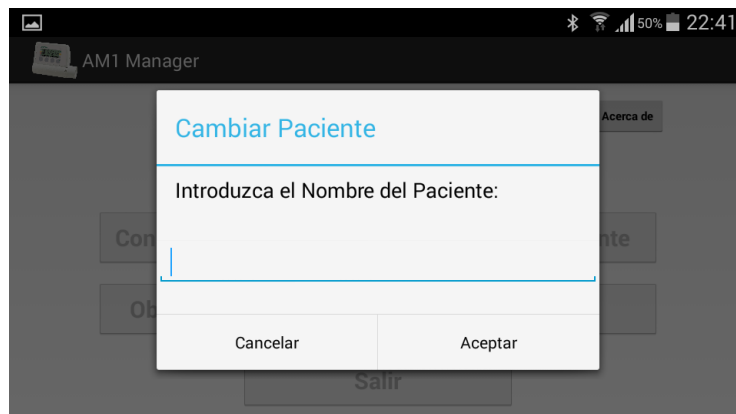


Figura A.4: Cuadro de diálogo para cambiar el nombre del paciente.

**Obtener Mediciones.** Para obtener las mediciones guardadas en la memoria del AM1 pulse “Obtener Mediciones”. Se desplegará un cuadro de diálogo como el de la figura A.5. Puede usar la barra de desplazamiento si no caben todas las medidas en la pantalla. Si desea visualizar una gráfica con los valores del parámetro PEF pulse “Gráfica PEF”; si no, pulse volver. Una vez obtenida la gráfica (figura A.6) puede regresar al menú principal pulsando la tecla atrás de su dispositivo.

Resultados Espirometría								
Nombre del Paciente: Carlos Contreras - N° Mediciones: 105								
FVC (ml)	FEP (l/min)	FEV1 (ml)	FEF75 (ml/s)	FEF50 (ml/s)	FEF25 (ml/s)	FEF7525 (ml/s)	Fecha	Hora
3173	220	2917	2261	2771	3587	2749	03.09.14	15.34
2232	143	1710	1513	1360	1649	1460	03.09.14	15.34
2551	184	2551	2890	2941	3043	2890	03.09.14	15.34
1813	199	1813	2958	3077	3315	3084	03.09.14	15.34
2654	217	2645	3026	3383	3468	3317	04.09.14	17.55

Figura A.5: Datos de las mediciones.

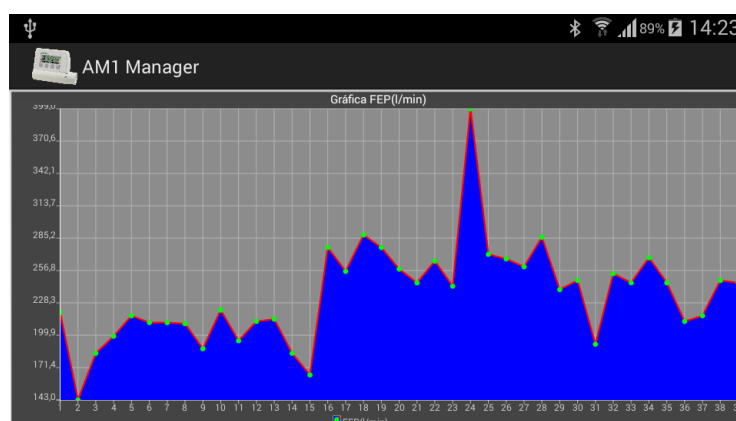


Figura A.6: Gráfica con los valores del parámetro PEF.

**Borrar Datos.** Para borrar la memoria del AM1 pulse “Borrar Datos” y responda afirmativamente al cuadro como el de la figura A.7.

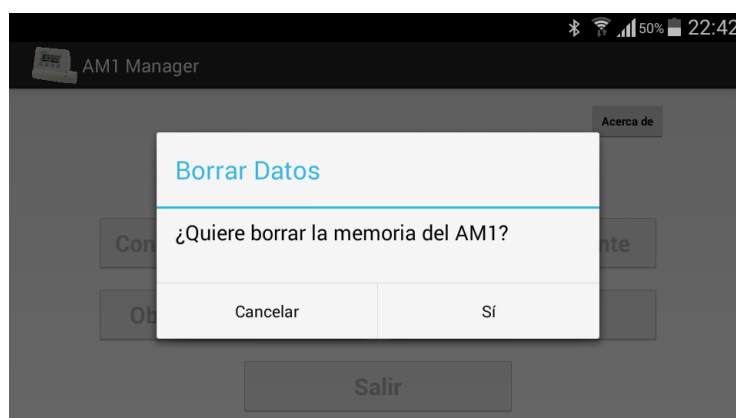


Figura A.7: Confirmación para borrar la memoria del espirómetro.

**Salir.** Para salir de la aplicación pulse “Salir” y responda afirmativamente al cuadro como el de la figura A.8. Esta acción también apagará el espirómetro.

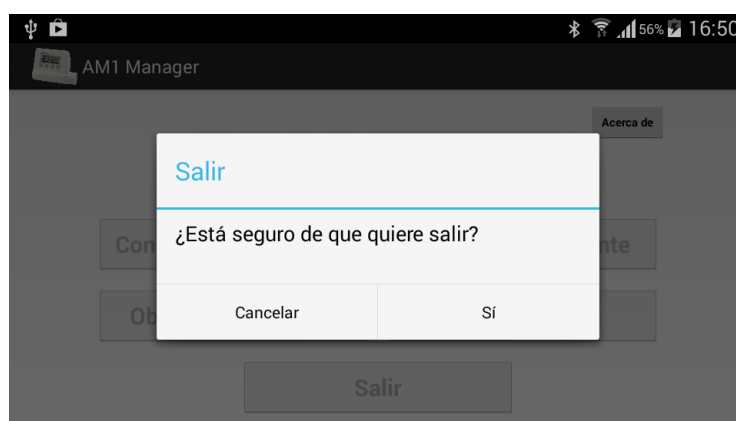


Figura A.8: Confirmación para salir de la aplicación.

**Error.** Si en cualquiera de las operaciones anteriores se produce un error, la aplicación lo notificará y volverá al estado inicial (figura A.9) para conectar de nuevo con el espirómetro.

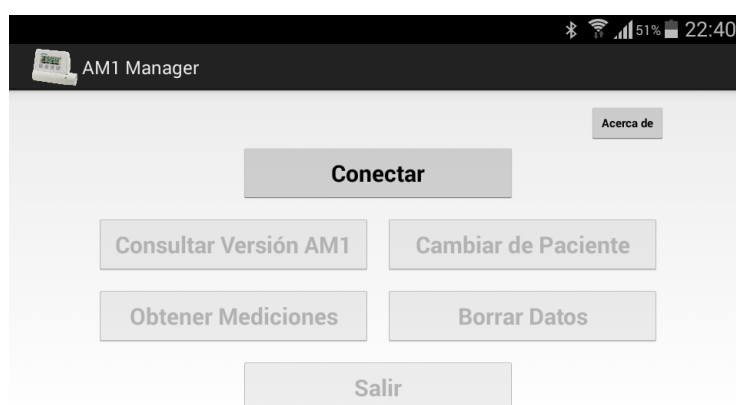


Figura A.9: Pantalla inicial de la aplicación.





# Bibliografía

- [1] Grupo de Bioingeniería y Telemedicina. Universidad Politécnica de Madrid. *Telemedicina y dispositivos inteligentes*. <http://www.gbt.tfo.upm.es/Telemedicina+y+dispositivos+inteligentes>.
- [2] El Hospital. Información para el desarrollo de la salud en América Latina. *¿Qué es la telemedicina?*. <http://www.elhospital.com/temas/Que-es-la-telemedicina+8082249?pagina=1>.
- [3] Archivos de Bronconeumología. *Espirometría*. <http://www.archbronconeumol.org/es/espirometria/articulo/90224086/>.
- [4] 20minutos.es. *España lidera en Europa en uso de ‘smartphones’ con un 66 % de tasa de penetración*. <http://www.20minutos.es/noticia/1900266/0/espana-lidera/uso-smartphones/66-penetracion/>.
- [5] El androide libre. *La historia y los comienzos de Android, el sistema operativo de Google*. <http://www.elandroidelibre.com/2011/08/la-historia-y-los-comienzos-de-android-el-sistema-operativo-de-google.html>.
- [6] BloombergBusinessweek *Google Buys Android for Its Mobile Arsenal*. <http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal>.

- [7] AndroidCurso. *Las versiones de Android y niveles de API*.  
<http://www.androidcurso.com/index.php/tutoriales-android/31-unidad-1-vision-general-y-entorno-de-desarrollo/146-las-versiones-de-android-y-niveles-de-api>.
- [8] Android Developers. *Bluetooth*. <http://developer.android.com/guide/topics/connectivity/bluetooth.html>.
- [9] McLibre.com. *¿Qué es el XML?*.  
[http://www.mclibre.org/consultar/xml/lecciones/xml\\_quees.html](http://www.mclibre.org/consultar/xml/lecciones/xml_quees.html).
- [10] Eclipse.org. *What is Eclipse?*. [http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fint\\_eclipse.htm](http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fint_eclipse.htm).
- [11] Página Oficial del IDE Eclipse. <http://www.eclipse.org>.
- [12] Spirometry. *Spirometry Tests*.  
<http://www.spirometry.guru/spirometry.html>.
- [13] Jesús Tomás Gironés. *El Gran Libro de Android Avanzado*, 1ª edición. MARCOMBO, S.A. ISBN: 9788426720788.
- [14] Salvador Gómez Oliver. *Manual de Programación Android*, 2011.  
<http://www.sgoliver.net>
- [15] Maestros del Web. *Curso Android: Desarrollo de aplicaciones móviles*, 2011.  
<http://www.maestrosdelweb.com/editorial/curso-android/>.
- [16] Universitat Politècnica de València. *Curso de Android*, 2014.  
<http://www.androidcurso.com/>.

- 
- [17] Universidad de Murcia. *Desarrollo de Aplicaciones para Dispositivos Móviles Android. Tema 5*. <http://universidad.informaticosmurcia.es/Descargas/Cursos/Android/parteII/II-Tema5.pdf>.
- [18] Android Developers. *Handler*.  
<http://developer.android.com/reference/android/os/Handler.html>.
- [19] Android Developers. *BroadcastReceiver*.  
<http://developer.android.com/reference/android/content/BroadcastReceiver.html#ReceiverLifecycle>.
- [20] “DebaterOfMath”. *Android development Tutorial Bluetooth*.  
<https://www.youtube.com/watch?v=0TQHZ16q0Ik>.
- [21] Daniel García. *Activando y desactivando el bluetooth en Android*.  
<http://danielggarcia.wordpress.com/2013/10/19/bluetooth-i-activando-y-desactivando-el-bluetooth-en-android/>.
- [22] Stack Overflow. *Simple Bluetooth data receiver Android*.  
<http://stackoverflow.com/questions/9164138/simple-bluetooth-data-receiver-android>.
- [23] Stack Overflow. *How do you set the max number of characters for an EditText in Android?* <http://stackoverflow.com/questions/6066212/how-do-you-set-the-max-number-of-characters-for-an-edittext-in-android>.
- [24] Stack Overflow. *Center two buttons horizontally*.  
<http://stackoverflow.com/questions/4189883/center-two-buttons-horizontally>.

- [25] Stack Overflow. *How to create dialog which will be full in horizontal dimension*. <http://stackoverflow.com/questions/11613825/how-to-create-dialog-which-will-be-full-in-horizontal-dimension>.
- [26] Jon Segador. *Diferentes unidades de medida disponibles en Android: dp, sp, pt, px, mm, in*. <http://jonsegador.com/2012/09/diferentes-unidades-de-medida-disponibles-en-android-dp-sp-pt-px-mm-in/>.
- [27] Androideity. *Layout en Android II: Relative Layout*. <http://androideity.com/2011/07/16/layout-en-android-relative-layout/>.
- [28] Androideity. *Trabajando con Threads en Android I: Messages*. <http://androideity.com/2011/09/15/trabajando-con-threads-en-android-i/>.
- [29] Developing Frogtek. *Usando estilos en Android*. <http://developing.frogtek.org/2010/07/16/usando-estilos-en-android/>.
- [30] Canvas html 5. *Android: Thread (Hilo) y Handler, proceso en segundo plano*. <http://www.tutorialeshtml5.com/2012/05/android-thread-hilo-y-handler-proceso.html>.
- [31] El Baúl del Programador. *Programación Android: Interfaz gráfica: Diálogos y notificaciones*. [http://elbauldelprogramador.com/programacion-android-interfaz-grafica\\_11/](http://elbauldelprogramador.com/programacion-android-interfaz-grafica_11/).
- [32] David Santo Orcero. Universidad de Málaga. Fundación Observatorio Universidad-Empresa de Málaga. *Procesamiento de textos y presentaciones con L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>*.

