

# Práctica con Bluetooth

Antonio J. Galán Herrera

---



## Introducción

Esta memoria pretende describir la elaboración de la práctica mencionada proporcionando una revisión de la realización de los ejercicios pedidos, la elaboración del código y un breve historial de la evolución del proyecto.

Las acciones más relevantes en esta práctica son el **descubrimiento de dispositivos**, la **búsqueda de servicios** y el **cliente** para el servidor Bluetooth, siendo esta última clase la más compleja de todo el proyecto, debido a que requiere de las dos anteriores para que funcione correctamente: el cliente debe buscar el servicio de chat al que conectarse entre otros conjuntos de servicios, donde dichos conjuntos de servicios pertenecen a los dispositivos encontrados mediante el descubrimiento de dispositivos.

---

<b>Introducción</b>	<b>1</b>
<b>Evolución de la práctica</b>	<b>3</b>
Planteamiento	3
Desarrollo	3
Resolución	3
<b>Disposición de las clases</b>	<b>4</b>
DispComBluetooth	4
InfoLocal	4
DescDispRemotos	4
DescubrirDispositivos	4
DescServRemotos	5
BuscarServicios	5
RFCOMM	5
Cliente	5
Servidor	6
Utiles	6
Filtro	6
ServicioBasico	6
<b>Funcionamiento de la aplicación</b>	<b>7</b>
Servidor	7
Cliente	7

---

## Evolución de la práctica

### Planteamiento

Desde un primer momento pensé en crear tantos paquetes como apartados tuviese la práctica, donde en cada uno añadiría una clase **EjercicioX** para cada ejercicio que tuviera el apartado en cuestión. Si un ejercicio dependía del código de otro, mi intención era crear una clase nueva con una copia del código base al que después le añadiría el nuevo contenido del ejercicio indicado.

### Desarrollo

Sin embargo, una vez leídos los enunciados y habiendo empezado a escribir un poco de código, me di cuenta de que esa era una manera bastante ineficiente. Entonces, decidí construir las clases conteniendo la mayor cantidad de ejercicios posibles y añadirle sentencias **if-else** con las que guiar al usuario hacia la función deseada (ejercicio), mientras se le informaba usando mensajes de la consola. Así lograba reducir el número de clases y además, aumentar la eficiencia de todo el proyecto.

### Resolución

Por último, para organizar más fácilmente las clases, las agrupé en paquetes con el mismo nombre que los apartados de la práctica: **DispComBluetooth** (Dispositivo de Comunicación Bluetooth), **DescDispRemotos** (Descubrimiento de Dispositivos Remotos), **DescServRemotos** (Descubrimiento de Servicios Remotos) y **RFCOMM** (Comunicación a través de RFCOMM de Bluetooth); además del paquete **Utiles** para clases de soporte para los ejercicios.

---

## Disposición de las clases

A continuación se muestran todas las clases del proyecto, clasificadas por los distintos paquetes en las que se encuentran.

### DispComBluetooth

Contiene el ejercicio 1.

#### InfoLocal

*Devuelve un objeto con información del dispositivo local.*

Esta clase define un objeto que contiene el nombre, la dirección Bluetooth, el tipo y la versión de radio del dispositivo local, posee un constructor, métodos de obtención de los datos anteriores y un método **toString** con el que poder visualizar la información.

### DescDispRemotos

Contiene los ejercicios 2 y 3.

#### DescubrirDispositivos

*Ejecuta una búsqueda de dispositivos Bluetooth y muestra por pantalla el nombre y la dirección Bluetooth de los dispositivos encontrados.*

Esta clase ejecuta un proceso para descubrir dispositivos y almacena los encontrados en una lista de tipo **ArrayList**, que será usada por la clase **BuscarServicios** para encontrar los servicios indicados en cada elemento de la lista.

Esta clase posee un objeto **Filtro** encargado de verificar el cumplimiento de las condiciones por las que un dispositivo debe filtrarse: escribiendo «si» se activa, pidiendo al usuario un nombre y/o dirección Bluetooth del dispositivo que busca, puede dejarse un campo vacío para ignorar ese atributo en el filtrado; escribiendo «no», el filtro no se instancia y por tanto, se hace el descubrimiento de dispositivos por defecto (buscar todos).

---

## DescServRemotos

Contiene los ejercicios 4, 5, 6 y 7.

### BuscarServicios

*Ejecuta una búsqueda de servicios en cada dispositivo que haya sido descubierto.*

Esta clase ejecuta un proceso de búsqueda de servicios sobre cada elemento de una lista de dispositivos y devuelve un objeto **ServicioBasico** si un filtro ha sido activado antes de dicha búsqueda de servicios.

Esta clase posee un objeto **Filtro** encargado de verificar el cumplimiento de las condiciones por las que un servicio debe filtrarse: escribiendo «si» se activa, pidiendo al usuario un nombre y/o dirección URL del servicio que busca, puede dejarse un campo vacío para ignorar ese atributo en el filtrado; escribiendo «no», el filtro no se instancia y por tanto, se hace la búsqueda de servicios por defecto (buscar todos).

Esta clase además, contiene un método interno **arreglar()** que dado un **String** elimina el último byte (0) para que pueda usarse adecuadamente por el método **equals()**.

## RFCOMM

Contiene la aplicación cliente-servidor.

### Cliente

*Ejecuta un proceso de conexión a un servidor como cliente.*

Esta clase permite un intercambio secuencial de mensajes de texto con un servidor proporcionado.

Se encarga de ejecutar la clase **BuscarServicios** para encontrar el servicio de chat al que conectarse entre todos los servicios de todos los dispositivos encontrados anteriormente. Una vez obtenido dicho servicio, se conecta usando su URL al **Servidor**.

---

## Servidor

*Ejecuta un proceso de conexión a un cliente como servidor.*

Esta clase permite un intercambio secuencial de mensajes de texto con un cliente que se conecte.

Esta clase ya viene proporcionada para probar el cliente, que es el auténtico ejercicio de la práctica, pero merece la pena incluirla ya que un cliente sin un servidor al que conectarse no tiene mucho sentido.

## Utiles

Contiene clases de soporte para una mayor modularización del proyecto.

## Filtro

*Comprueba si un dispositivo Bluetooth o un servicio cumple con las condiciones especificadas por el usuario para ser filtrado o no.*

Esta clase hace una correcta verificación de los valores a los campos pedidos en los métodos, comprobando previamente qué valor o valores debe evaluar: `valorA` y `valorB`, únicamente el `valorA` o únicamente el `valorB`.

## ServicioBasico

*Almacena el nombre y la dirección URL de un servicio.*

Esta clase cuenta con un constructor y dos métodos: uno para obtener el nombre del servicio y otro para obtener la dirección URL.

La clase tendrá una instancia solo si se activa un **Filtro** en la clase **BuscarServicios**, de esta forma el **Cliente** obtiene la información necesaria del servicio filtrado a través de esta clase, permitiéndole obtener la dirección URL y conectarse al **Servidor** indicado.

---

## Funcionamiento de la aplicación

Todos los datos necesarios son mostrados por la consola en el inicio del Servidor y del Cliente, salvo en el emulador, que requiere añadir el código del Servidor y del Cliente dentro de su propia clase.

### Servidor

Lo primero y más importante es que debe ejecutarse el Servidor antes que el Cliente, para que se genere un servicio con la dirección URL del Servidor a la que poder conectarse.

1. Ejecución en el mismo dispositivo que el Cliente.
  - 1.1. Instalar la librería del emulador.
  - 1.2. Ejecutar el emulador del Servidor.
2. Ejecución en un dispositivo distinto al del Cliente.
  - 2.1. Ejecutar la clase Servidor del proyecto.

### Cliente

Ahora ya sí, suponiendo que el Servidor haya arrancado y esté esperando peticiones, se deben ejecutar 2 filtros.

1. Ejecutar la clase Cliente.
2. Filtrar el dispositivo donde se aloja el Servidor.
3. Filtrar el servicio de chat del dispositivo filtrado.
4. La conexión será automática si se encuentra el servicio.

#### NOTAS:

*Se trata de una aplicación Bluetooth, por lo que ambos dispositivos deben encontrarse uno cerca del otro y con la función de Bluetooth activada.*

*En algunos casos es necesario que los dispositivos estén emparejados.*

---

## Referencias

### Bluecove

Página principal: <http://www.bluecove.org/>

Estándar los UUIDs: <https://www.bluetooth.org/en-us/specification/assigned-numbers>

Protocolo SDP: <https://www.bluetooth.org/en-us/specification/assigned-numbers/service-discovery>

### Repositorio de GitHub

<https://github.com/15Galan/asignatura-810/tree/master/Tema%202/Pr%C3%A1cticas/Pr%C3%A1ctica%20Bluetooth>