Redes Inalámbricas

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.

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 las 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 una lista objetos de tipo **`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).

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 obtener una lista de servicios y conectarse a uno de ellos. Una vez obtenido dicho servicio, se conecta usando su URL al **`Servidor`**.

La función principal de esta clase en la práctica es la de encontrar el servicio de chat y conectarse a él.

ClienteGUI

Ejecuta un proceso de conexión a un servidor como cliente.

Esta clase a diferencia que la clase `Cliente`, posee una interfaz gráfica simple.

Sin embargo, tiene las mismas funciones que la clase mencionada, a excepción de poder mostrar por la consola información adicional de la ventana como su activación, desactivación, minimización, etc...

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, abriendo el servicio sobre una dirección URL y mediante un UUID de SerialPort (SPP).

El nombre del servicio de chat es «Chat», descrito en la URL en el código de la clase.

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: *nombre* y *dirección*, únicamente el *nombre* o únicamente la *dirección*. Ambos valores se comparan usando **`equalsIgnoreCase()`**.

Dependiendo de la clase que cree el filtro, el *nombre* puede ser el de un servicio o el de un dispositivo, mientras que la *dirección* será la dirección URL si se escribió un servicio o la dirección Bluetooth si se escribió un dispositivo.

Para ignorar un campo específico en un filtro, basta con dejarlo en blanco pulsando Enter sin haber escrito nada en él, esto hará que ese valor sea **`null`** y más tarde en la verificación de los valores, este no se compruebe.

ServicioSimple

Almacena el nombre y la dirección URL de un servicio, así como el dispositivo que lo ofrece.

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. Además, si hubiese más de un dispositivo ofreciendo un servicio con el mismo nombre, el método **`toString()`** de esta clase proporcionaría información adicional, diferenciando los servicios por dispositivos.

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.

En este caso, se simulará la conexión del cliente a un servidor que ofrece un chat.

Servidor

Lo primero y más importante es que debe ejecutarse el servidor antes que el ciente, 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 en el proyecto (ya lo está).
   1.2. Ejecutar la clase del emulador de la clase `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 debe activar al menos, el filtro de servicios.

1. Filtrar (o no) el dispositivo donde se aloja el servidor.
2. Filtrar (o no) por servicios de tipo SerialPort Profile (SPP).
3. Filtrar el servicio de chat para almacenarlo en la lista.
4. **Conectarse al servicio buscado.**
   4.1. La conexión será automática si se encuentra un solo servicio.
   4.2. La conexión será manual (indicando el ID) si se encuentra más de un servicio.

NOTA: Se trata de una aplicación Bluetooth, por lo que ambos dispositivos deben encontrarse uno cerca del otro y con la función de Bluetooth activada, ambos dispositivos en modo «visible».

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

Ver la evolución de todo el proyecto:
https://github.com/15Galan/asignatura-810/tree/master/Tema%202/Pr%C3%A1cticas/Pr%C3%A1ctica%20Bluetooth