Programación Bluetooth en Java

1. **Utiliza la documentación de la API y los métodos de la clase *LocalDevice* para obtener información acerca de la configuración de tu dispositivo Bluetooth local ¿Qué información puedes obtener?**

```java
import javax.bluetooth.*;

public class ej1 {

    Run | Debug
    public static void main(String[] args) {
        try {
            LocalDevice ld = LocalDevice.getLocalDevice();

            System.out.println("Direccion del dispositivo Bluetooth: " + ld.getBluetoothAddress());
            System.out.println("Nombre del dispositivo Bluetooth: " + ld.getFriendlyName());
            System.out.println("Encendido dispositivo Bluetooth: " + ld.isPowerOn());
            System.out.println(x:"Propiedades del dispositivo Bluetooth: ");
            System.out.println("\t* Version: " + ld.getProperty("bluecove"));
            System.out.println("\t* Conecciones activas: " + ld.getProperty("bluecove.connections"));

            // RemoteDevice dev = RemoteDevice.getRemoteDevice(con);
        } catch (BluetoothStateException e) {
            System.err.println(e.toString());
        }
    }
}
```

En mi caso, se puede mostrar información como la dirección bluetooth del dispositivo, su nombre, si está encendido y algunas propiedades.

```
BlueCove version 2.1.1-SNAPSHOT on winsock
Direccion del dispositivo Bluetooth: D0ABD5914B68
Nombre del dispositivo Bluetooth: DESKTOP-IP8U1UN
Encendido dispositivo Bluetooth: true
Propiedades del dispositivo Bluetooth:
        * Version: 2.1.1-SNAPSHOT
        * Conecciones activas: 0
BlueCove stack shutdown completed
PS G:\My Drive\Universidad\Redes Inalámbricas>
```

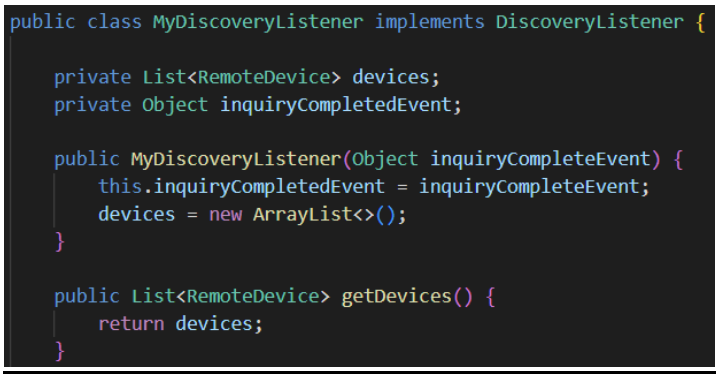
2. Implementación de una aplicación que permita descubrir todos los dispositivos bluetooth visibles cercanos. Muestra su nombre y su dirección Bluetooth.

Usaremos 2 clases de java, una clase principal donde declarar las clases necesarias y una clase *MyDiscoveryListener*, donde vamos a sobrescribir algunos métodos de la clase *DiscoveryListener*.

(En la captura se puede ver la explicación de cada una de las partes)

```
Run | Debug
public static void main(String[] args) {

    Object inquiryCompleteEvent = new Object();

    try {
        // El descubrimiento de dispositivos consta de 3 pasos
        // 1. Referencia al objeto local (proporciona acceso y control al dispositivo
        // local)
        LocalDevice ld = LocalDevice.getLocalDevice();

        // 2. Obtener referencia de donde esta mi dispositivo local
        DiscoveryAgent da = ld.getDiscoveryAgent();

        // 3. Utilizar el DiscoveryAgent para encontrar dispositivos cercanos
        DiscoveryListener listener = new MyDiscoveryListener(inquiryCompleteEvent);

        // Contamos el tiempo que tarda en buscar dispositivos
        long initTime = System.nanoTime();

        // Se inicia la busqueda de dispositivos
        // GIAC: General Inquiry access Code - (General –buscamos dispositivos visibles
        // para todos)
        // LIAC: Limited Inquiry access Code) –(Limitado –visibles sólo para los que
        // conocemos)
        boolean start = da.startInquiry(DiscoveryAgent.GIAC, listener);

        // USAMOS SEMAFOROS - MONITORES para controlar el fin de la busqueda
        synchronized (inquiryCompleteEvent) {
            if (start) {
                try {
                    System.out.println(x:"Buscando dipositivos...");
                    inquiryCompleteEvent.wait();
                    long endTime = System.nanoTime();
                    System.out.println(x:"Fin inquiry.");
                    System.out.println("Tiempo busqueda: " + (endTime - initTime) / 1e9 + "segundos.");
                } catch (Exception e) {
                    // TODO: handle exception
                }
            }
        }
    } catch (BluetoothStateException e) {
        System.err.println(e.toString());
    }
}
```

Dentro de la clase *MyDiscoveryListener* se usa una lista para almacenar los dispositivos encontrados y un objeto *inquiryCompletedEvent* (junto con un monitor) para sincronizar la clase con la clase principal.



```
public class MyDiscoveryListener implements DiscoveryListener {

    private List<RemoteDevice> devices;
    private Object inquiryCompletedEvent;

    public MyDiscoveryListener(Object inquiryCompleteEvent) {
        this.inquiryCompletedEvent = inquiryCompleteEvent;
        devices = new ArrayList<>();
    }

    public List<RemoteDevice> getDevices() {
        return devices;
    }
```

Una vez hemos encontrado un dispositivo, el método *deviceDiscovered* nos informa, mostrando por pantalla la dirección bluetooth y su nombre, y guarda el dispositivo en la lista de dispositivos.

```
@Override
public void deviceDiscovered(RemoteDevice arg0, DeviceClass arg1) {
    try {
        System.out.println("Dispositivo encontrado: " + arg0.getBluetoothAddress() + " " + arg0.getFriendlyName(true));
        devices.add(arg0);
    } catch (Exception e) {
        // TODO: handle exception
    }
}
```

Posteriormente, una vez que la búsqueda de dispositivos ha finalizado, el método *inquireCompleted* notifica al programa principal (libera el hilo con notifyAll) para que la ejecución del objeto en la clase principal continue.

```
@Override
public void inquiryCompleted(int arg0) {
    System.out.println(x:"Busqueda terminada. ");
    synchronized (inquiryCompletedEvent) {
        inquiryCompletedEvent.notifyAll();
    }
}
```

Si hacemos una búsqueda…

```
PS G:\My Drive\Universidad\Redes Inalámbricas>  & 'C:\Program Fil
BlueCove version 2.1.1-SNAPSHOT on winsock
Buscando dipositivos...
Dispositivo encontrado: F4939FA953FF Wireless Controller
Dispositivo encontrado: C097270531D9 [TV] Samsung 6 Series (49)
Dispositivo encontrado: 9C19C23B5712 MI AIRDOTS BASIC_R
Dispositivo encontrado: E09F2A1C8E4C Intuos BT S
Dispositivo encontrado: FC58FAD6A0C7 LG FJ0(C7)
Busqueda terminada.
Fin inquiry.
Tiempo busqueda: 43.7793169segundos.
BlueCove stack shutdown completed
PS G:\My Drive\Universidad\Redes Inalámbricas>
```

3. **Implementación de una aplicación que permita descubrir un dispositivo concreto (por dirección Bluetooth y/o friendly name).**

```
Object inquiryCompleteEvent = new Object();
String movil = "Mi 10";

try {
    // El descubrimiento de dispositivos consta de 3 pasos
    // 1. Referencia al objeto local (proporciona acceso y control al dispositivo
    // local)
    LocalDevice ld = LocalDevice.getLocalDevice();

    // 2. Obtener referencia de donde esta mi dispositivo local
    DiscoveryAgent da = ld.getDiscoveryAgent();

    // 3. Utilizar el DiscoveryAgent para encontrar dispositivos cercanos
    DiscoveryListener listener = new MyDiscoveryListener(inquiryCompleteEvent, movil);
```

Creamos un objeto (String movil) en la clase principal, que pasaremos por el objeto “listener” para la clase *MyDiscoveryListener*, modificando a la vez el constructor de este, permitiendo que el método *deviceDiscovered* busque los dispositivos con el nombre de la variable “movil” o por otro lado por su dirección Bluetooth.

```java
@Override
public void deviceDiscovered(RemoteDevice arg0, DeviceClass arg1) {
    try {
        if (movil.equals(arg0.getFriendlyName(true)) || movil.equals(arg0.getBluetoothAddress())) {
            System.out.println(
                    "Dispositivo encontrado: " + arg0.getBluetoothAddress() + " " + arg0.getFriendlyName(true));
            devices.add(arg0);
            this.inquiryCompleted(arg0:0);
        }

    } catch (Exception e) {
        // TODO: handle exception
    }
}
```

```
BlueCove version 2.1.1-SNAPSHOT on winsock
Buscando dipositivos...
Dispositivo encontrado: C097270531D9 [TV] Samsung 6 Series (49)
Busqueda terminada.
Fin inquiry.
Tiempo busqueda: 40.0893209segundos.
BlueCove stack shutdown completed
```

4. Implementación de una aplicación que permita descubrir todos los servicios (de clase público) de todos los dispositivos cercanos.

Para buscar los servicios necesitamos primero buscar los dispositivos, si hemos encontrado algún dispositivo, estarán guardados en la lista de dispositivos remotos.

```java
if (!devices.isEmpty()) {
    // A partir de una lista de dispositivos detectados se buscan los servicios de
    // una determinada clase
    UUID uuids[] = new UUID[1];
    uuids[0] = new UUID(0x1002); // servicios ofrecidos publicamente

    int attridset[] = new int[2];
    attridset[0] = SERVICE_NAME_ATTRID;
    attridset[1] = SERVICE_ID_ATTRID;

    for (RemoteDevice rd : devices) {

        System.out.println("Servicios activos del dispositivo: " + rd.getBluetoothAddress()
                + ":" + rd.getFriendlyName(true));
        da.searchServices(attridset, uuids, rd, listener);
        synchronized (inquiryCompleteEvent) {
            inquiryCompleteEvent.wait();
        }
    }
}
```

Encontrados los dispositivos, declaramos los UUIDs (en nuestro solo será 1, el 0x1002, que es para mostrar servicios públicos), enteros que hacen referencia al nombre de servicio e ID y mostraremos una lista de todos los dispositivos y sus servicios con el siguiente método de la clase *MyDiscoveryListener*.

```java
@Override
public void serviceSearchCompleted(int arg0, int arg1) {
    System.out.println(x:"Busqueda de servicios terminada. ");
    synchronized (inquiryCompletedEvent) {
        inquiryCompletedEvent.notifyAll();
    }

}
```

Una vez la búsqueda finaliza, informa a la clase principal que la

búsqueda de servicios ha finalizado.

El método *servicesDiscovered* recibe una lista de servicios. Con esos servicios buscamos obtener las URLs de los mismos con el método *getConnectionURL*.

Si la URL no es null, la añadimos a la lista y conseguimos el nombre del servicio, si es que lo tiene. Si lo tiene, mostramos por pantalla el nombre y la URL del servicio. Si no, solo mostramos la URL.

```java
@Override
public void servicesDiscovered(int arg0, ServiceRecord[] arg1) {
    for (ServiceRecord service : arg1) {

        String url = service.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);

        if (url == null) {
            continue;
        }

        urlServices.add(url);

        if (service.getAttributeValue(0x0100) != null) {
            System.out.println("Servicio " + service.getAttributeValue(0x0100).getValue());
            System.out.println("URL: " + url);
        } else {
            System.out.println("Servicio: " + url);
        }
    }
}
```

```
BlueCove version 2.1.1-SNAPSHOT on winsock
Buscando dipositivos...
Dispositivo encontrado: 5444A3EF0F85 [TV] Samsung BU8505 55 TV
Dispositivo encontrado: F4939FA953FF Wireless Controller
Dispositivo encontrado: C097270531D9 [TV] Samsung 6 Series (49)
Dispositivo encontrado: 9C19C2385712 MI AIRDOTS BASIC_R
Dispositivo encontrado: 983B8FD84310 DESKTOP-4UK9A55
Dispositivo encontrado: E09F2A1C8E4C Intuos BT S
Dispositivo encontrado: FC58FAD6A0C7 LG FJ0(C7)
Busqueda terminada.
Fin inquiry.
Tiempo busqueda: 74.0144891segundos.
Dispositivos encontrados: 7
Servicios activos del dispositivo: 5444A3EF0F85:[TV] Samsung BU8505 55 TV
Servicio: btl2cap://5444A3EF0F85:001f;authenticate=false;encrypt=false;master=false
Servicio: btl2cap://5444A3EF0F85:001f;authenticate=false;encrypt=false;master=false
Servicio Samsung Smart TV Audio
URL: btl2cap://5444A3EF0F85:0017;authenticate=false;encrypt=false;master=false
Servicio Advanced Audio Source
URL: btl2cap://5444A3EF0F85:0019;authenticate=false;encrypt=false;master=false
Servicio: btl2cap://5444A3EF0F85:0017;authenticate=false;encrypt=false;master=false
Servicio Advanced Audio Sink
URL: btl2cap://5444A3EF0F85:0019;authenticate=false;encrypt=false;master=false
Servicio Headset Gateway
URL: btspp://5444A3EF0F85:2;authenticate=false;encrypt=false;master=false
Servicio Handsfree Gateway
URL: btspp://5444A3EF0F85:3;authenticate=false;encrypt=false;master=false
Servicio :1.285
URL: btspp://5444A3EF0F85:4;authenticate=false;encrypt=false;master=false
Servicio :1.223
URL: btspp://5444A3EF0F85:5;authenticate=false;encrypt=false;master=false
Servicio :1.296
URL: btspp://5444A3EF0F85:6;authenticate=false;encrypt=false;master=false
Servicio: btl2cap://5444A3EF0F85:001f;authenticate=false;encrypt=false;master=false
Servicio :1.344
URL: btspp://5444A3EF0F85:7;authenticate=false;encrypt=false;master=false
Servicio :1.227
URL: btspp://5444A3EF0F85:8;authenticate=false;encrypt=false;master=false
Busqueda de servicios terminada.
Servicios activos del dispositivo: F4939FA953FF:Wireless Controller
Busqueda de servicios terminada.
Servicios activos del dispositivo: C097270531D9:[TV] Samsung 6 Series (49)
Servicio: btl2cap://C097270531D9:001f;authenticate=false;encrypt=false;master=false
Servicio: btl2cap://C097270531D9:001f;authenticate=false;encrypt=false;master=false
Servicio Samsung Smart TV Audio
```

La lista de servicios, dada la cantidad de dispositivos encontrados es muy larga, muestro alguno de los muchos servicios.

5. Implementación de una aplicación que permita descubrir todos los servicios (de clase público) de un dispositivo concreto (especificado por dirección Bluetooth y/o friendly name).

Aplicamos la misma idea que el ejercicio 4, solo que esta vez, al igual que en el ejercicio 3, declaramos un objeto de tipo String con el nombre (o dirección Bluetooth) del dispositivo que queramos encontrar y la pasamos por el constructor de la clase *MyDiscoveryListener*.

```
BlueCove version 2.1.1-SNAPSHOT on winsock
Buscando dipositivos...
Dispositivo encontrado: C097270531D9 [TV] Samsung 6 Series (49)
Busqueda terminada.
Fin inquiry.
Tiempo busqueda: 79.6768512segundos.
Dispositivos encontrados: 1
Servicios activos del dispositivo: C097270531D9:[TV] Samsung 6 Series (49)
Servicio: btl2cap://C097270531D9:001f;authenticate=false;encrypt=false;master=false
Servicio: btl2cap://C097270531D9:001f;authenticate=false;encrypt=false;master=false
Servicio Samsung Smart TV Audio
URL: btl2cap://C097270531D9:0017;authenticate=false;encrypt=false;master=false
Servicio Advanced Audio Source
URL: btl2cap://C097270531D9:0019;authenticate=false;encrypt=false;master=false
Servicio: btl2cap://C097270531D9:0017;authenticate=false;encrypt=false;master=false
Servicio :1.0000000000000000299
URL: btspp://C097270531D9:2;authenticate=false;encrypt=false;master=false
Servicio :1.0000000000000000210
URL: btspp://C097270531D9:3;authenticate=false;encrypt=false;master=false
Servicio :1.0000000000000000209
URL: btspp://C097270531D9:4;authenticate=false;encrypt=false;master=false
Busqueda de servicios terminada.
BlueCove stack shutdown completed
```

6. Implementación de una aplicación que permita descubrir un servicio con un nombre concreto en un dispositivo concreto.

Primero, debemos añadir un nuevo String (String service) a la clase *MyDiscoveryListener* para guardar el servicio a buscar y añadirlo al constructor.

```java
@Override
public void servicesDiscovered(int arg0, ServiceRecord[] arg1) {
    for (ServiceRecord service : arg1) {

        String url = service.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);

        if (url == null) {
            continue;
        }

        urlServices.add(url);

        if (service.getAttributeValue(0x0100) != null) {
            String name_service = (String) service.getAttributeValue(0x0100).getValue();
            if (name_service.equals(this.service)) {
                System.out.println("Servicio: " + service.getAttributeValue(0x0100).getValue());
                System.out.println("URL: " + url);
                urlServices.add(url);
            }
        }
    }
}
```

En el método *servicesDiscovered* comprobamos si el servicio descubierto es el que estamos buscando. Si lo es, mostramos por pantalla el servicio en cuestión y la URL de este, y lo añadimos a la lista de URLs.

7. Implementa una aplicación cliente-servidor que permita el envío de mensajes de texto entre ellos (como un chat textual). Ambos procesos (cliente y servidor) son procesos Java distintos que se ejecutan en máquinas diferentes para comprobar la comunicación inalámbrica.

Extendiendo lo dicho en el ejercicio 4, una vez encontrada la URL del servicio “chat”, creamos una conexión, mostramos la dirección Bluetooth y nombre del servidor y comenzamos el intercambio de mensajes hasta que el servidor nos responda con “FIN”.

```java
if (!listener.getUrlServices().isEmpty()) {
    // URL del destino
    String url = listener.getUrlServices().get(index:0);
    System.out.println(url);
    // Inicio de conexion
    StreamConnection conexion = (StreamConnection) Connector.open(url);
    RemoteDevice dev = RemoteDevice.getRemoteDevice(conexion);
    System.out.println("Direccion Bluetooh del servidor: " + dev.getBluetoothAddress());
    System.out.println("Nombre del servidor: " + dev.getFriendlyName(false));

    InputStream is = conexion.openInputStream();
    BufferedReader br = new BufferedReader(new InputStreamReader(is));

    OutputStream os = conexion.openOutputStream();
    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(os));

    Scanner sc = new Scanner(System.in);
    String response = "";
    String mensaje = "";

    while (!response.equals(anObject:"FIN.")) {

        System.out.print(ld.getFriendlyName() + ": ");
        response = sc.nextLine();
        bw.write(response);
        bw.newLine();
        bw.flush();
        System.out.println(x:"Esperano respuesta del servidor...");
        mensaje = br.readLine();
        System.out.println(
                dev.getBluetoothAddress() + " - " + dev.getFriendlyName(true) + ": "
                        + mensaje);
    }
    System.out.println(
            "Dispositivo " + dev.getBluetoothAddress() + " - " + dev.getFriendlyName(true)
                    + " desconectado correctamente");
    br.close();
    bw.close();
    sc.close();
    mensaje = "";
    conexion.close();
}
```

Ejemplo de comunicación entre dos dispositivos Cliente – Servidor:

```
BlueCove version 2.1.1-SNAPSHOT on winsock
Buscando dipositivos...
Dispositivo encontrado: 983B8FD84310 DESKTOP-4UK9A55
Busqueda terminada.
Fin inquiry.
Tiempo busqueda: 63.5036864segundos.
Dispositivos encontrados: 1
Servicios activos del dispositivo: 983B8FD84310:DESKTOP-4UK9A55
Busqueda de servicios terminada.
btspp://983B8FD84310:4;authenticate=false;encrypt=false;master=false
Direccion Bluetooh del servidor: 983B8FD84310
Nombre del servidor: DESKTOP-4UK9A55
DESKTOP-IP8U1UN: hola chaval
Esperano respuesta del servidor...
983B8FD84310 - DESKTOP-4UK9A55: adios nene
DESKTOP-IP8U1UN:
```

Cliente

```
BlueCove version 2.1.1-SNAPSHOT on winsock
Datos del servidor: 983B8FD84310 - DESKTOP-4UK9A55

Servidor Iniciado. Esperando clientes...
Direcci?nn del dispositivo remoto: D0ABD5914B68
Nombre del dispositivo remoto: DESKTOP-IP8U1UN
Esperano respuesta del cliente...
D0ABD5914B68 - DESKTOP-IP8U1UN: hola chaval
DESKTOP-4UK9A55: adios nene
Esperano respuesta del cliente...
```

Servidor

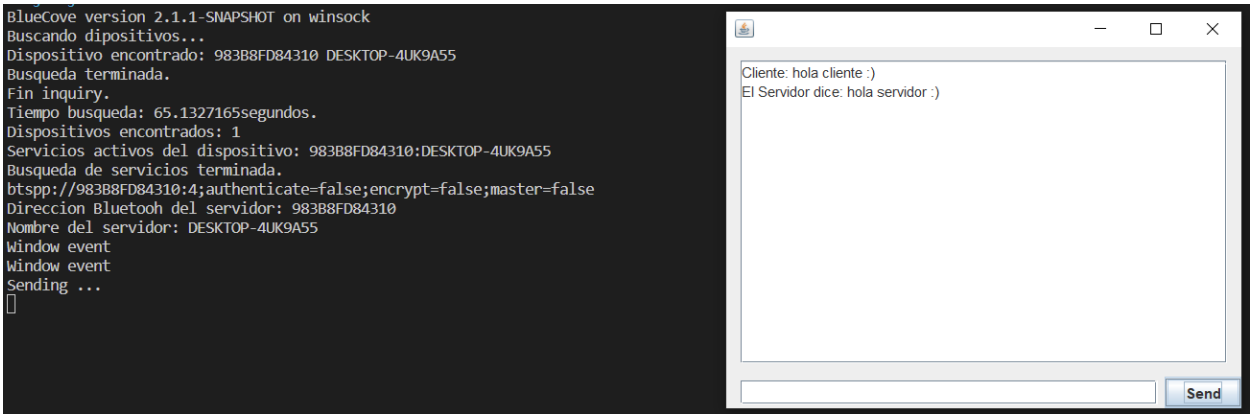
8. **Ejercicio opcional: una vez que tengas desarrollada la funcionalidad del cliente, integra su funcionalidad (y la del servidor) con una interfaz gráfica para probar la comunicación entre ambos interlocutores.**

Implementaremos un objeto *chatWindow* para que tanto el cliente como el servidor muestren su conversación en una ventana emergente y no por consola.

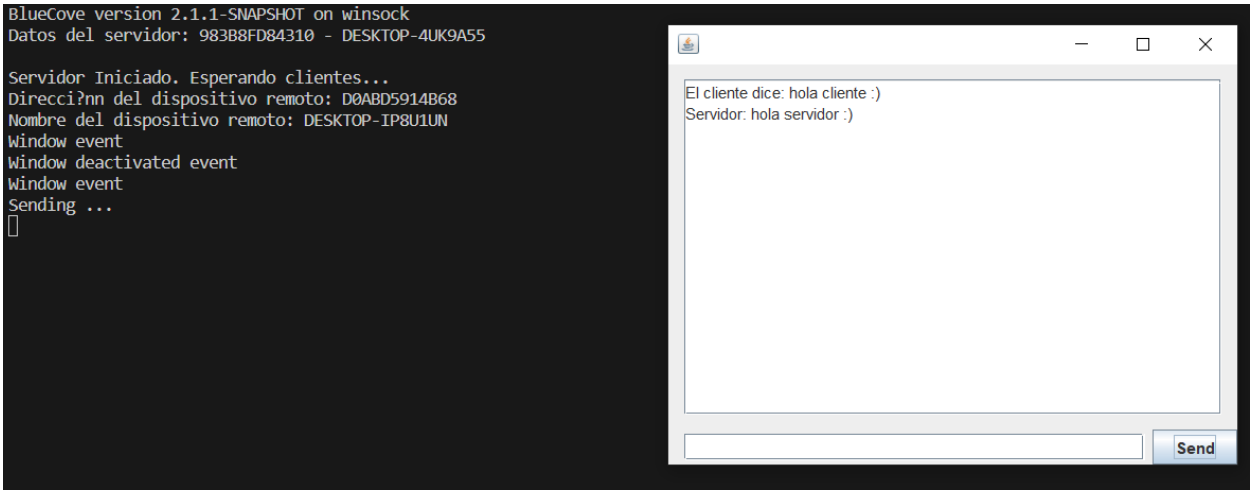
```
/*
 * Invocamos la ventana (que se ejecuta como un thread en segundo plano) y
 * definimos la acci□n de enviar lo que insertemos por teclado
 */
final chatUI.ChatWindow _window;
_window = new ChatWindow();
_window.setVisible(b:true);
_window.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {

        /* MODIFICAR EL C□DIGO PARA EL ENV□O AQU□ */
        try {
            String response = _window.getIn(); // metodo que lee de la entrada
            _window.setOut("Cliente: " + response); // Metodo que escribe en la salida
                                                    // de la ventana
            bw.write(response);
            bw.newLine();
            bw.flush();
        } catch (Exception i) {
            i.printStackTrace();
        }

    }
});

/*
 * L□neas obligatorias: hay que registrar un listener para los eventos de
 * ventana y
 * sobre el de window closing, realizar el cierre de conexione
 */

_window.addWindowListener(new java.awt.event.WindowListener() {
    public void windowClosing(WindowEvent e) {
        System.out.println(x:"Window closing event .... close connections");
    }

    public void windowClosed(WindowEvent e) {
        System.out.println(x:"Window closed event ");
    }

    public void windowDeactivated(WindowEvent e) {
        System.out.println(x:"Window deactivated event ");
    }

    public void windowOpened(WindowEvent e) {
        System.out.println(x:"Window event ");
    }

    public void windowIconified(WindowEvent e) {
        System.out.println(x:"Window event ");
    }

    public void windowDeiconified(WindowEvent e) {
        System.out.println(x:"Window event ");
    }

    public void windowActivated(WindowEvent e) {
        System.out.println(x:"Window event ");
    }

});

/*
 * En este punto, una vez iniciada la ventana, nos ponemos en bucle a
 * recibir la informaci□n del otro extremo
 */
/* MODIFICAR EL C□DIGO PARA LA RECEPCI□N AQU□ */
String mensaje = "";
while (!mensaje.equals(anObject:"FIN.")) {
    mensaje = br.readLine();
    _window.setOut("El Servidor dice: " + mensaje);
}
_window.setOut("Servidor successfully disconnected from the server");
br.close();
bw.close();
conexion.close();
```

Implementación necesaria tanto para el Cliente como para el Servidor

Un ejemplo de ejecución seria:



Cliente



Servidor