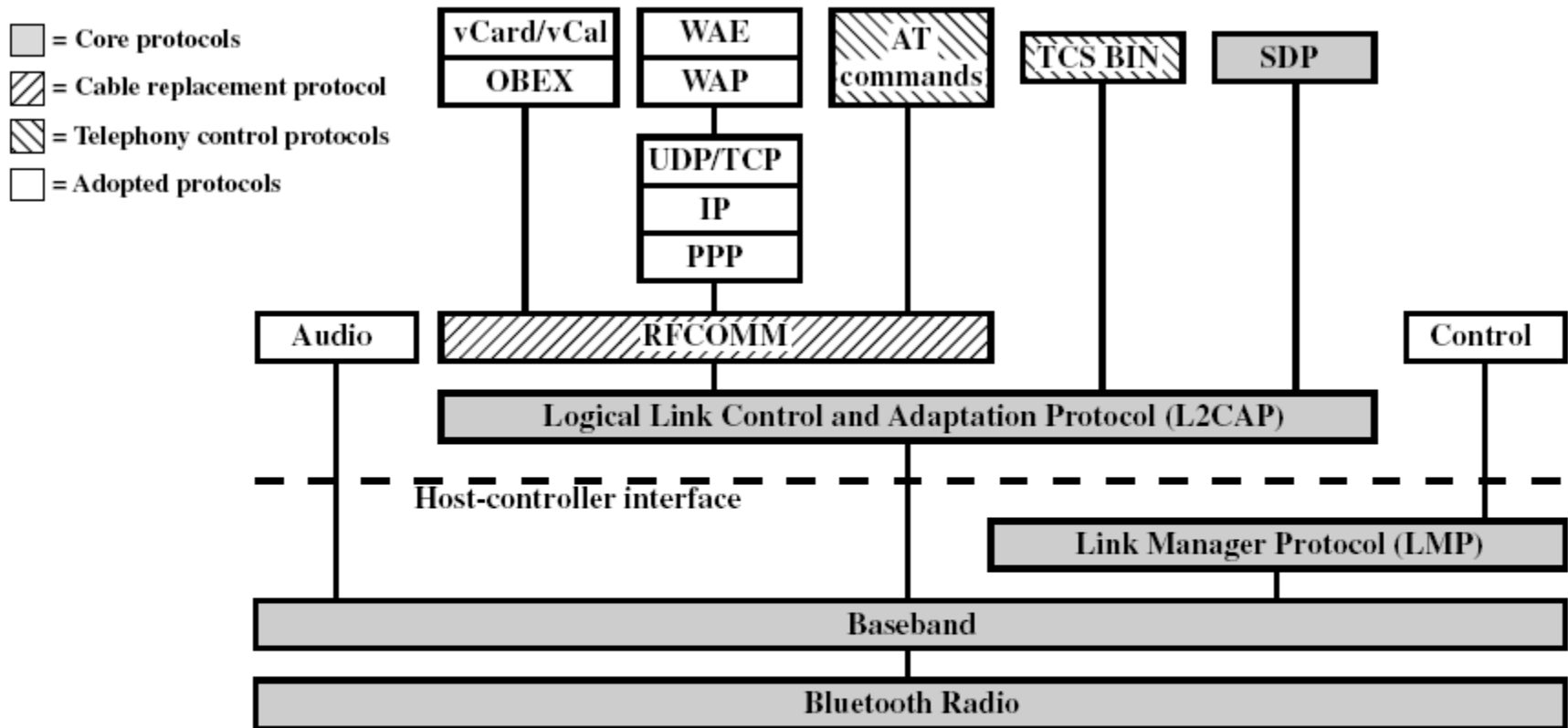




Bluetooth

Programación de aplicaciones

Pila de protocolos de Bluetooth



AT	= attention sequence (modem prefix)	TCS BIN	= telephony control specification - binary
IP	= Internet Protocol	UDP	= User Datagram Protocol
OBEX	= Object exchange protocol	vCal	= virtual calendar
PPP	= Point-to-Point Protocol	vCard	= virtual card
RFCOMM	= radio frequency communications	WAE	= wireless application environment
SDP	= service discovery protocol	WAP	= wireless application protocol
TCP	= transmission control protocol		

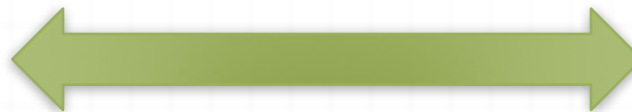
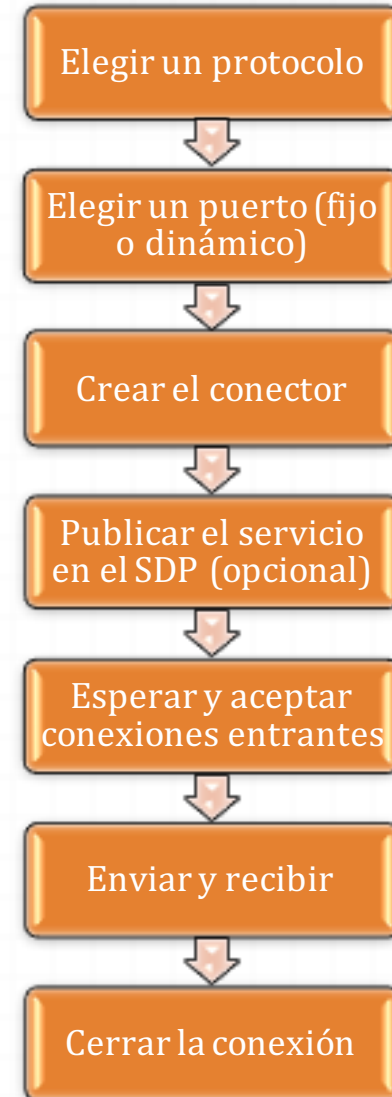
Programación de Aplicaciones Bluetooth

- 0 ¿cómo crear programas que conecten dos dispositivos Bluetooth?
 - 0 El proceso para establecer una conexión dependerá si el dispositivo en cuestión está estableciendo una conexión de entrada (incoming) o de salida (outgoing)
 - 0 Los dispositivos que inician una conexión de salida necesitan elegir un dispositivo destino (target) y un protocolo de transporte antes de establecer una conexión y transferir datos.
 - 0 Los dispositivos que establecen una conexión de entrada necesitan elegir un protocolo de transporte, escuchar antes de aceptar una petición y transferir datos

Conexiones salientes



Conexiones entrantes



Seleccionando un dispositivo

- 0 Cada chip Bluetooth lleva incorporado de fábrica un identificador único de 48 bits, denominado dirección Bluetooth o dirección de dispositivo (Bluetooth address- device address)
 - 0 Es la única forma de direccionamiento en Bluetooth
 - 0 Es posible proporcionar un identificador “user-friendly”
 - 0 Ej. “my phone”, “my car”, ...
- 0 Búsqueda de dispositivos en la proximidad
 - 0 Es el proceso de búsqueda y detección de dispositivos en la vecindad.
 - 0 DEVICE INQUIRY

Descubrimiento de dispositivos

Device Discovery



- 0 El dispositivo que pregunta transmite mensajes de pregunta (inquiry message) en 32 de los 79 canales Bluetooth.
- 0 El dispositivo Bluetooth descubrible (discoverable) escucha periódicamente estos mensajes y responde a los mensajes
- 0 Cada respuesta incluye la dirección Bluetooth del dispositivo y un entero que identifica la clase de dispositivo
 - 0 Móvil
 - 0 PC
 - 0 Auriculares



... en la práctica

- 0 **Conectable:** un dispositivo con Bluetooth dentro del alcance que responderá a otro dispositivo y establecerá una conexión.
- 0 **Modo visible:** cuando un dispositivo Bluetooth está en "modo visible", otros dispositivos Bluetooth pueden detectarlo y enlazarse o conectarse a él.
 - 0 Los dispositivos normalmente desactivan el modo visible después de un tiempo.
 - 0 **Encontrar dispositivo:** proceso que permite a un dispositivo detectar otros dispositivos *visibles*.
 - 0 **Enlazar:** proceso de creación de un vínculo persistente entre dos dispositivos Bluetooth, que puede incluir el intercambio de claves entre ambos. Este proceso solo ocurre una vez; toda conexión futura entre los dispositivos se autenticará automáticamente.

Bluetooth

- 0 La única forma de que un dispositivo detecte a otro es iniciar el proceso de descubrimiento de dispositivos
- 0 Detectabilidad vs conectividad
 - 0 Por cuestiones de ahorro de energía y privacidad, todos los dispositivos Bluetooth tienen dos opciones (estados) que determinan si el dispositivo responderá o no a las peticiones de detección (descubrimiento) y de conexión.
 - 0 Inquiry Scan
 - 0 Controla las peticiones de descubrimiento
 - 0 ON es descubrible
 - 0 Page Scan
 - 0 Controla las peticiones de conexión
 - 0 ON acepta peticiones de conexión entrantes

Inquiry Scan	Page Scan	Interpretación: El dispositivo ...
✓	✓	Es detectable y acepta peticiones de conexión entrantes (DEFECTO)
×	✓	No es detectable y acepta peticiones de conexión entrantes realizadas por dispositivos que tenían previamente su dirección (DEFECTO)
✓	×	Es detectable pero no acepta peticiones de conexión entrantes
×	×	No es detectable y no acepta peticiones de conexión entrantes. El dispositivo sólo establece conexiones de salida.

Conexiones. Protocolo de Transporte

- 0 Bluetooth proporciona 4 protocolos de transporte
 - 0 RFCOMM
 - 0 Radio Frequency Communications
 - 0 Protocolo orientado a flujo (similar a TCP)
 - 0 L2CAP
 - 0 Logical Link Control and Adaptation Protocol
 - 0 Protocolo orientado a paquete que puede configurarse con varios niveles de fiabilidad
 - 0 La configuración será compartida por TODAS las conexiones L2CAP a un mismo dispositivo
 - 0 Sirve para encapsular conexiones RFCOMM

RFCOMM

L2CAP

ACL
(Banda Base)

SCO
(Audio)

Conexiones. Protocolo de Transporte

- 0 Bluetooth proporciona 4 protocolos de transporte
 - 0 ACL
 - 0 Asynchronous Connection-oriented Logical protocol
 - 0 Encapsula la conexiones L2CAP
 - 0 SCO
 - 0 Synchronous Connection-Oriented protocol
 - 0 Protocolo basado en paquetes que trasmite exclusivamente audio con calidad de voz (64kb/s)
 - 0 Es no fiable → los paquetes no se retransmiten

RFCOMM

L2CAP

ACL
(Banda Base)

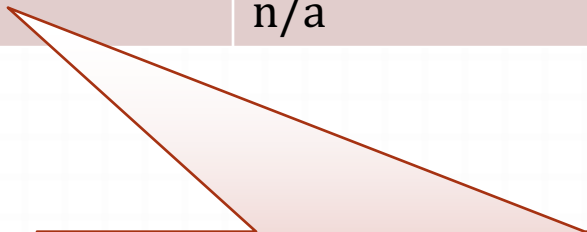
SCO
(Audio)

Bluetooth: Puertos

- 0 Un puerto se utiliza para permitir que varias aplicaciones ejecuten conexiones en el mismo dispositivo
- 0 En L2CAP los puertos se denominan *Protocol Service Multiplexers (PSM)* – Multiplexador de Servicios de Protocolos
 - 0 Toman valores IMPARES entre 1 y 32.767
- 0 En RFCOMM los puertos se denominan canales (*channels*) estando disponibles del 1-30
- 0 L2CAP reserva los puertos de 1-1023 para servicios estándar
 - 0 SDP usa el puerto 1
 - 0 Las conexiones RFCOMM se multiplexan por el puerto 3
 - 0 L2CAP solo transporta datos, no voz y todos los puertos son fiables

Puertos en Bluetooth

Protocolo	Término	Puertos reservados/conocidos	Puertos no reservados (libres)
RFCOMM	Canal	Ninguno	1-30
L2CAP	PSM	Impares 1-4095	Impares 4097-32.765
SCO	n/a	n/a	N/A

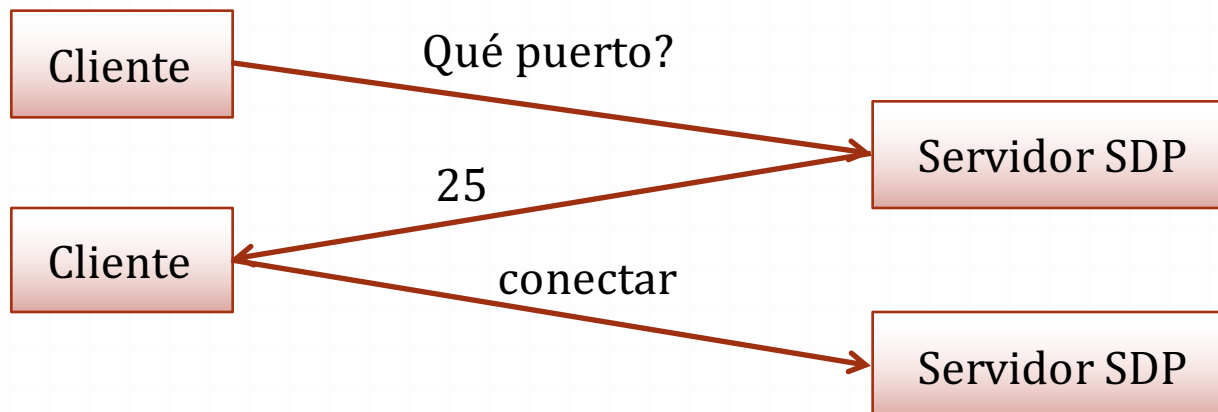


Dado que dos dispositivos pueden tener como máx. 1 conexión SCO entre ellos, no existe la noción de puerto en SCO

Protocolo de Descubrimiento de Servicios

SDP

- 0 Para conocer el puerto de conexión, se utiliza el SDP
- 0 Cada dispositivo Bluetooth mantiene un servidor SDP
- 0 Permite la asignación dinámica de puertos
 - 0 Cuando un servicio arranca, puede elegir cualquier puerto arbitrario que no esté en uso.



Descripción de Servicios

0 Registro de Servicio – Service Record

- 0 La descripción de un servicio que el servidor de la aplicación registra en su servidor SDP y que el servidor SDP transmite a los clientes se denomina Registro de Servicio.
- 0 Consiste en una lista de pares atributos /valor
- 0 Cada atributo es un entero de 16 bits
- 0 Cada valor puede ser de un tipo básico de datos (una cadena de caracteres o un entero)
- 0 Los dos atributos más importantes son:
 - 0 Service ID
 - 0 Service Class ID List

Service record

ServiceID
•0x1234-...

ServiceClassIDList
•0xABCD-...
•0x5678-...

Service Name
•"Example Service"

Protocol description List
•L2CAP
•RFCOMM
•Port 3

SDP

0 Service ID

- 0 Cada servicio tiene asignado un identificador único
- 0 El cliente conoce este identificador
- 0 El espacio de identificadores únicos válidos es mayor que el número de puertos
- 0 UUID – Identificadores Universales Únicos de 128 bits
- 0 Cada desarrollador elige el UUID del servicio en tiempo de diseño y se asocia al registro del servicio
- 0 Notación Estándar para un UUID
 - 0 Dígitos hexadecimales separados por guiones

XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXXX

SDP

0 Service Class ID List

- 0 Para servicios que proporcionan el mismo tipo de servicio se define un segundo UUID, denominado Service Class ID
 - 0 Dos programas diferentes pueden publicar el mismo Service Class ID
- 0 Para las aplicaciones que ofrecen varios servicios, Bluetooth permite que cada servicio tenga una lista de clases de servicios proporcionados

UUIDs reservados

- 0 Bluetooth tiene un conjunto de UUIDs reservados
 - 0 SHORT UUIDs
 - 0 Permiten identificar clases de servicios predefinidos, protocolos de transporte y perfiles
 - 0 Los 96 bits más bajos de los UUIDs reservados tienen el mismo valor, y se refieren por los 16 o 32 bits superiores
 - 0 Para obtener el UUID de 128 bits a partir de un short UUID (16 o 32 bits):

$$128_bit_UUID = 16_or_32_bit_number * 2^{96} + Bluetooth_Base_UUID$$

Atributos del SDP

Servicio	UUID reservado
SDP	0x0001
RFCOMM	0x0003
L2CAP	0x0100
SDP Service Class ID	0x1000
Serial Port Service Class ID	0x1101
Headset Service Class	0x1108

Bluetooth Base UUID = 00000000-0000-1000-8000-00805F9B34FB

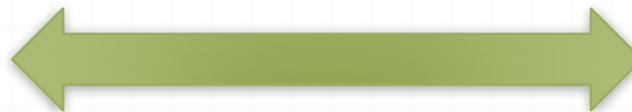
Atributos del SDP

- 0 Service Class ID List
- 0 Service ID
- 0 Service Name
- 0 Service Description
- 0 Protocol Description List
- 0 Profile Descriptor List
- 0 Service Record Handle

Conexiones salientes



Conexiones entrantes



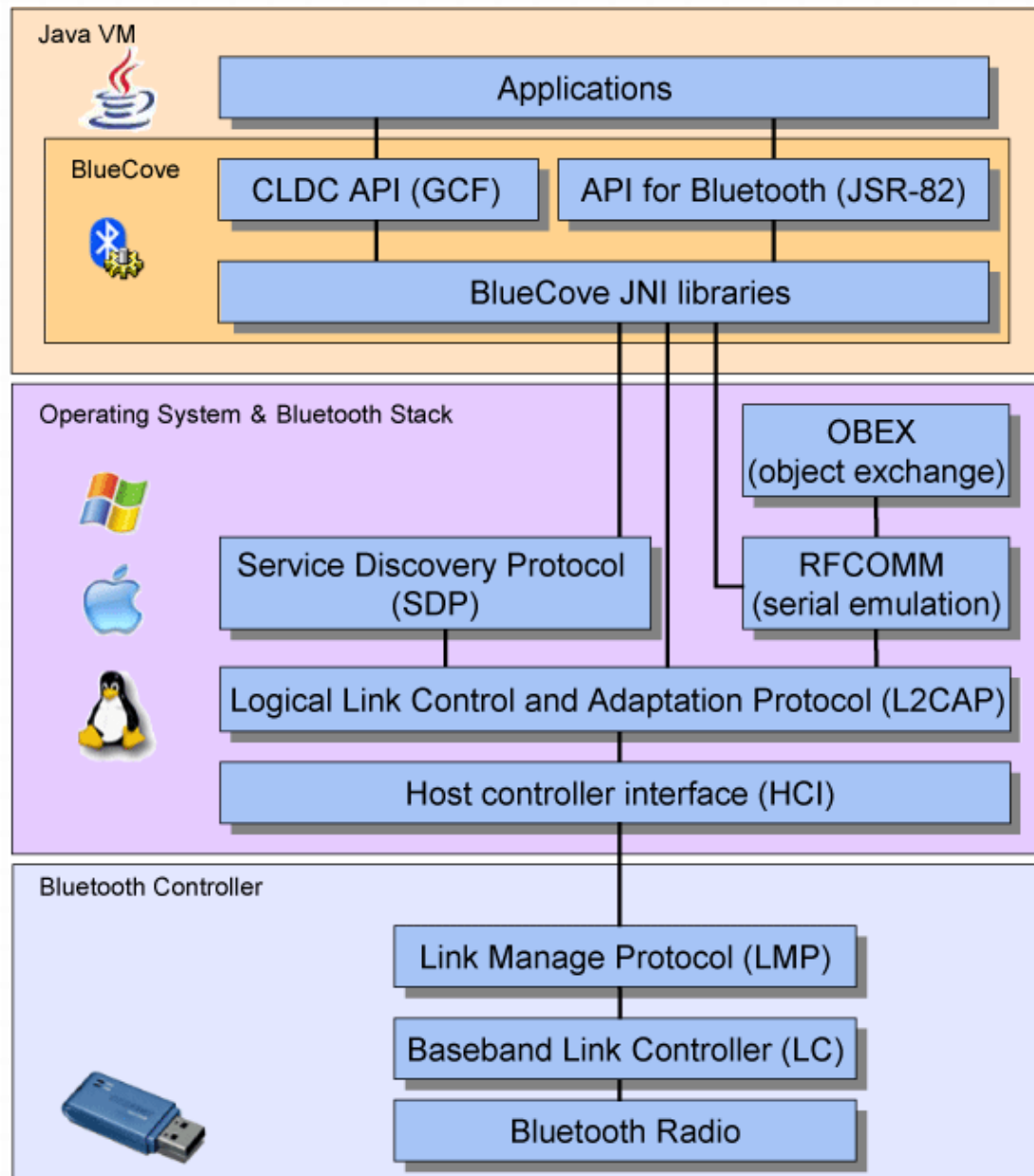
Java-JSR 82 BlueCove

<http://bluecove.org/>

Introducción a la programación Bluetooth

- 0 Java JSR-82 es la especificación Java para la programación de aplicaciones Bluetooth.
- 0 BlueCove es colección de librerías Java 2 SE para bluetooth que implementan algunas interfaces de la especificación JSR-82.
 - 0 BlueCove es de código abierto (opensource) y de libre distribución.
 - 0 Actualmente admite los siguientes perfiles
 - 0 SDAP – Service Discovery Application Profile
 - 0 RFCOMM – Serial Cable Emulation Protocol
 - 0 L2CAP – Logical Link Control And Adaptation Protocol
 - 0 OBEX – *Generic Object Exchange Profile.*





Conseguir la librería BlueCove



<http://bluecove.org/>



Documentación:

<http://www.bluecove.org/bluecove/apidocs/>

0 Distribución de bluecove:

<https://code.google.com/p/bluecove/>

Incluir [bluecove-2.1.0.jar](#) en el proyecto Java

0 Para 64 bits:

<http://snapshot.bluecove.org/distribution/download/2.1.1-SNAPSHOT/2.1.1-SNAPSHOT.62/>

Incluir [bluecove-2.1.1-SNAPSHOT.jar](#) en el proyecto Java





Características generales

- 0 La clase **`javax.bluetooth.RemoteDevice`** representa un dispositivo
 - 0 Proporciona información acerca de un dispositivo, incluyendo la dirección y el nombre del dispositivo
 - 0 Identificación de dispositivos
 - 0 Cada dispositivo Bluetooth tiene una dirección de 48 bits global única, denominada *dirección bluetooth* o dirección de dispositivo (**Bluetooth address – device address**) –
 - 0 Cada dispositivo tiene un nombre de dispositivo “user-friendly” denominado *nombre bluetooth* (**display name**)

Características generales

0 Métodos de javax.bluetooth.RemoteDevice

```
RemoteDevice rd;  
rd.getBluetoothAddress();  
rd.getFriendlyName();
```

0 Métodos de javax.bluetooth.LocalDevice

```
LocalDevice ld = LocalDevice.getLocalDevice();  
DiscoveryAgent da = ld.getDiscoveryAgent();
```

0 Métodos de DiscoveryAgent

```
da.startInquiry();  
Da.searchServices();
```

Características generales

- 0 Cada dispositivo Bluetooth tiene una dirección de 48 bits global única, denominada *dirección bluetooth* o dirección de dispositivo (Bluetooth address – device address) –
 - 0 asignada por el fabricante del dispositivo
 - 0 Utilizada en cualquier nivel o capa del proceso de comunicación Bluetooth.
 - 0 Método `String getAddress()`
 - 0 Cadena de 12 caracteres
- 0 Cada dispositivo tiene un nombre de dispositivo “user-friendly” denominado *nombre bluetooth (display name)*
 - 0 Asignado por el usuario ej. “My phone”
 - 0 Son arbitrarios y pueden estar duplicados
 - 0 Mostrado al usuario en lugar de la dirección bluetooth
 - 0 Método `String getFriendlyName(boolean alwaysAsk)`

Búsqueda de dispositivos cercanos – device inquiry

0 El descubrimiento de dispositivos consta de tres pasos:

1) Obtener una referencia al (único) objeto de la clase `LocalDevice`

0 Proporciona acceso y control al dispositivo bluetooth local

```
LocalDevice ld = LocalDevice.getLocalDevice();
```

2) Obtener una referencia al `DiscoveryAgent` del `LocalDevice`

```
DiscoveryAgent da = ld.getDiscoveryagent();
```


Búsqueda de dispositivos cercanos – device inquiry

3) Utilizar el DiscoveryAgent para comenzar descubrimiento de dispositivos a través del método

```
startInquiry(int accessCode, DiscoveryListener listener);
```

- 0 Dos códigos de acceso:

- 0 **GIAC** - General Inquiry access Code - (General – visibles para todos)

- 0 **LIAC** – Limited Inquiry access Code) – (Limitado – visible sólo para los que me conocen)

- 0 Interfaz DiscoveryListener

```
MyDiscoveryListener lstnr = MyDiscoveryListener();  
da.startInquiry(Discoveryagent.GIAC, lstnr);
```

Interface `javax.bluetooth.DiscoveryListener`

Método	Invocado cuando ...
<code>deviceDiscovered</code>	se encuentra un nuevo dispositivo durante el descubrimiento de dispositivo
<code>inquiryCompleted</code>	finaliza la búsqueda
<code>servicesDiscovered</code>	se encuentran servicio(s) durante la búsqueda de servicios
<code>serviceSearchCompleted</code>	Se completa la búsqueda de dispositivos o se finaliza debido a un error

Inquiry Scan	Page Scan	Interpretación: El dispositivo ...
✓	✓	Es detectable y acepta peticiones de conexión entrantes (DEFECTO)
×	✓	No es detectable y acepta peticiones de conexión entrantes realizadas por dispositivos que tenían previamente su dirección (DEFECTO)
✓	×	Es detectable pero no acepta peticiones de conexión entrantes
×	×	No es detectable y no acepta peticiones de conexión entrantes. El dispositivo sólo establece conexiones de salida.

Descubrimiento de servicios - SDP

- 0 La búsqueda de servicios utiliza las mismas clases que el descubrimiento de dispositivos:
 - 0 En JSR-82 no hay funciones específicas para la búsqueda de servicios de dispositivos en las cercanías, por lo que debe de realizarse específicamente buscando uno por uno.
 - 0 Las clases que representan los servicios y sus descripción son:
 - 0 ServiceRecord
 - 0 DataElement
 - 0 UUID :entero sin signo de 16-bit (UUID)
 - 0 El valor de un atributo es un objeto de la clase DataElement.
 - 0 Hay un conjunto de atributos de servicio que siempre se recuperan (o incluyen) en el ServiceRecord del servicio
 - 0 Service Record Handle, Service Class ID List, Service Record State, Service ID, Protocol Descriptor List

Descubrimiento de servicios - SDP

- 0 La clase `ServiceRecord` es una estructura de datos que proporciona varios métodos de acceso muy útiles.

`getConnectionURL`

Devuelve una cadena conteniendo la dirección Bluetooth y el puerto para establecer posteriormente la conexión

`getAttributeValue`

Accede a los elementos almacenado en una instancia de la clase `ServiceRecord`

- 0 A partir de una lista de dispositivos detectados se itera para cada uno de los dispositivos detectados la búsqueda de servicios publicados públicamente (UUID 0x1002)
- 0 La búsqueda de servicios en un dispositivo se realiza con el método `searchServices` del `DiscoveryAgent`

```
UUID uuids[] = new UUID[1];
Uuids[0]= new UUID (0x1002);
Int attridset[] = new int[1];
Attridset[0] = SERVICE_NAME_ATTRID; //0x0100
da.searchServices(attridset, uuids, remotedevice,
    lstnr);
```

```
.....
```

```
public void servicesDiscovered(int transID,
    ServiceRecord[] rec) {
    for (int i=0; i<rec.length;i++) {
        DataElement d = rec[i].getAttributeValue
            (SERVICE_NAME_ATTRID );
        if (d!=null)
            System.out.println((String)g.getValue());
        else System.out.println("Unnamed service");

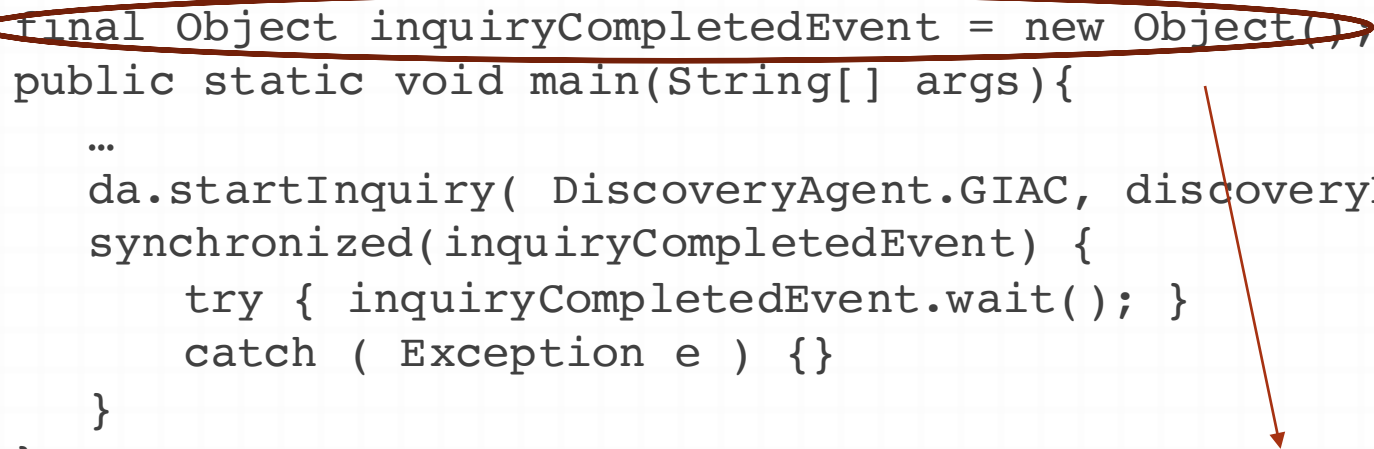
        System.out.println(rec[i].getURLConnection(ServiceR
            ecord.NOAUTHENTICATE_NOENCRYPT,false));
    }
}
```

Ejecución concurrente y asíncrona

- 0 Tanto el descubrimiento de dispositivos como servicios.
- 0 **Recomendación:** uso de semáforos o monitores para controlar el fin del proceso de descubrimiento.

1) En el proceso principal paramos el hilo de ejecución hasta que la búsqueda no se complete. Ejemplo:

```
final Object inquiryCompletedEvent = new Object();  
public static void main(String[] args){  
    ...  
    da.startInquiry( DiscoveryAgent.GIAC, discoveryListener);  
    synchronized(inquiryCompletedEvent) {  
        try { inquiryCompletedEvent.wait(); }  
        catch ( Exception e ) {}  
    }  
}
```



Podría sustituirse por un semáforo o por la clase del main (this) si implementa DiscoveryListener

Ejecución concurrente y asíncrona

2) En el `DiscoveryListener` avisamos al proceso principal cuando la búsqueda de dispositivos o servicios termine.

0 Modificamos el cuerpo de las funciones **`inquiryCompleted()`** y **`serviceSearchCompleted()`**, respectivamente. Ejemplo:

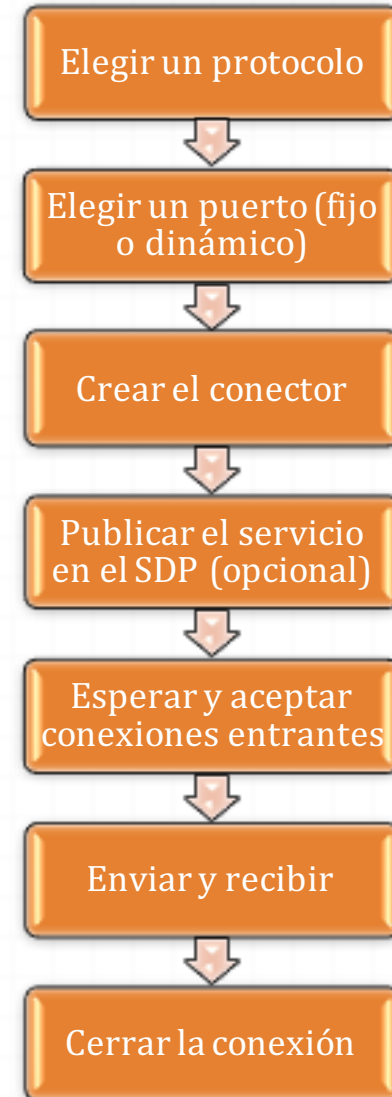
```
synchronized( inquiryCompletedEvent ) {  
    try { inquiryCompletedEvent.notifyAll();  
        } catch ( Exception e ) {}  
}
```


Programación de Sockets Bluetooth

Conexiones salientes



Conexiones entrantes



Sockets RFCOMM

0 Establecimiento de conexiones entrantes (Servidor)

1. Definir una URL que especifica datos necesarios para

1. Escuchar peticiones

2. Anunciar el servicio (Definir el ServiceRecord)

0 Estructura de la URL para RFCOMM

“btspp://” + “dirección Bluetooth del adaptador local”+“:” + “Service Class ID” + parámetros opcionales

0 Hasta 5 parámetros opcionales de la forma ;“attribute=value”

0 Name. Nombre de servicio

0 Authenticate (false). Si true, los dispositivos clientes deben autenticarse. Default: false

0 Encrypt (false). Si true, la conexión está encriptada

0 Authorize (false). Si true se requiere autorización (base de datos o usuario) antes de aceptar peticiones entrantes.

0 Master (false). El adaptador local es maestro en la pico red (*piconet*) asociada a las conexiones entrantes

2. La clase StreamConnectionNotifier es equivalente a socket de servidor y es necesaria para aceptar peticiones de conexión entrantes

```
StreamConnectionNotifier service =  
    (StreamConnectionNotifier) Connector.open(url);
```

Sockets RFCOMM

4. El servicio espera y acepta peticiones con el método `acceptAndOpen` sobre este objeto

```
StreamConnection con=(StreamConnection) service.acceptAndOpen();
```

5. Cuando se acepta una petición devuelve un objeto de la clase `StreamConnection` que se utiliza para intercambiar información con el dispositivo Bluetooth remoto.

0 Establecimiento de conexiones salientes (Cliente)

1. Construir una URL del servicio al que se quiere conectar
Manual o extraída del `ServiceRecord` del servicio

0 Tres parámetros opcionales

- 0 Master (false). El dispositivo cliente se convierte en maestro de la piconet
- 0 Encrypt
- 0 authenticate

2. Utilizar la URL para establecer una conexión a través del método (estático) `open` de la clase `Connector`.

```
StreamConnection con = (StreamConnection) Connector.open(url);
```

Uso de un socket RFCOMM conectado

- 0 Hay que obtener instancias de las clases `OutputStream` e `InputStream` para enviar y recibir datos a través de la conexión respectivamente

```
OutputStream os = con.openOutputStream();
```

```
InputStream is = con.openInputStream();
```

- 0 Para enviar datos → pasar un array de bytes al método `write` del objeto `OutputStream`

- 0 Devuelve los bytes enviados

```
String mensaje = "hola";
```

```
int bytes_sent = os.write(mensaje.getBytes());
```

- 0 Para recibir datos → para un array de bytes vacío al método `read` del objeto `InputStream`.

- 0 Devuelve el número de el nº de bytes leídos con éxito

```
byte read_buffer[] = new byte[80];
```

```
int bytes_received = is.read (read_buffer);
```

- 0 La conexión se cierra con el método `close`

```
con.close();
```

Algunas consideraciones ...

- 0 Existen varias implementaciones de stacks (pila de protocolos) Bluetooth. BlueCove es dependiente de la pila de protocolos que tenga implementada el sistema, limitando o ampliando así las opciones propias de JSR-82.
- 0 BlueCove es compatible con la Bluetooth Stack de Microsoft (winsock), que es la que garantizará el correcto funcionamiento de la aplicación, ya que es el dispositivo bluetooth utilizado es compatible con esta pila.
- 0 Limitaciones en la pila de protocolo de Windows:
 - 0 Microsoft Bluetooth Stack solo soporta conexiones RFCOMM.
 - 0 No soporta L2CAP
 - 0 `DiscoveryListener.deviceDiscovered()` debería ser llamado para dispositivos emparejados con la pila de protocolos de Microsoft a pesar de que el dispositivo este apagado o encendido.
 - 0 `LocalDevice.setDiscoverable(NOT_DISCOVERABLE)` no cambiará el estado del dispositivo si está en modo DISCOVERABLE (puede ser visto por otros dispositivos). Esta opción únicamente se podrá modificar desde la configuración del propio dispositivo bluetooth y no desde el propio código.
 - 0 No está implementado la autenticación, autorización y la encriptación, provista normalmente por el objeto `RemoteDevice` del JSR-82.

Limitaciones de la API BlueCove en:

<https://code.google.com/p/bluecove/wiki/stacks>