

Programmazione J2ME

Lezione 9

Il Bluetooth



Cosa impareremo oggi?

- L'API JSR-82 per la comunicazione tramite tecnologia Bluetooth
- Utilizzo dei protocolli RFCOMM e L2CAP

In questa lezione non ci addentreremo nei dettagli tecnici della tecnologia Bluetooth, ma vedremo solo come utilizzarla in un'applicazione J2ME

La tecnologia Bluetooth

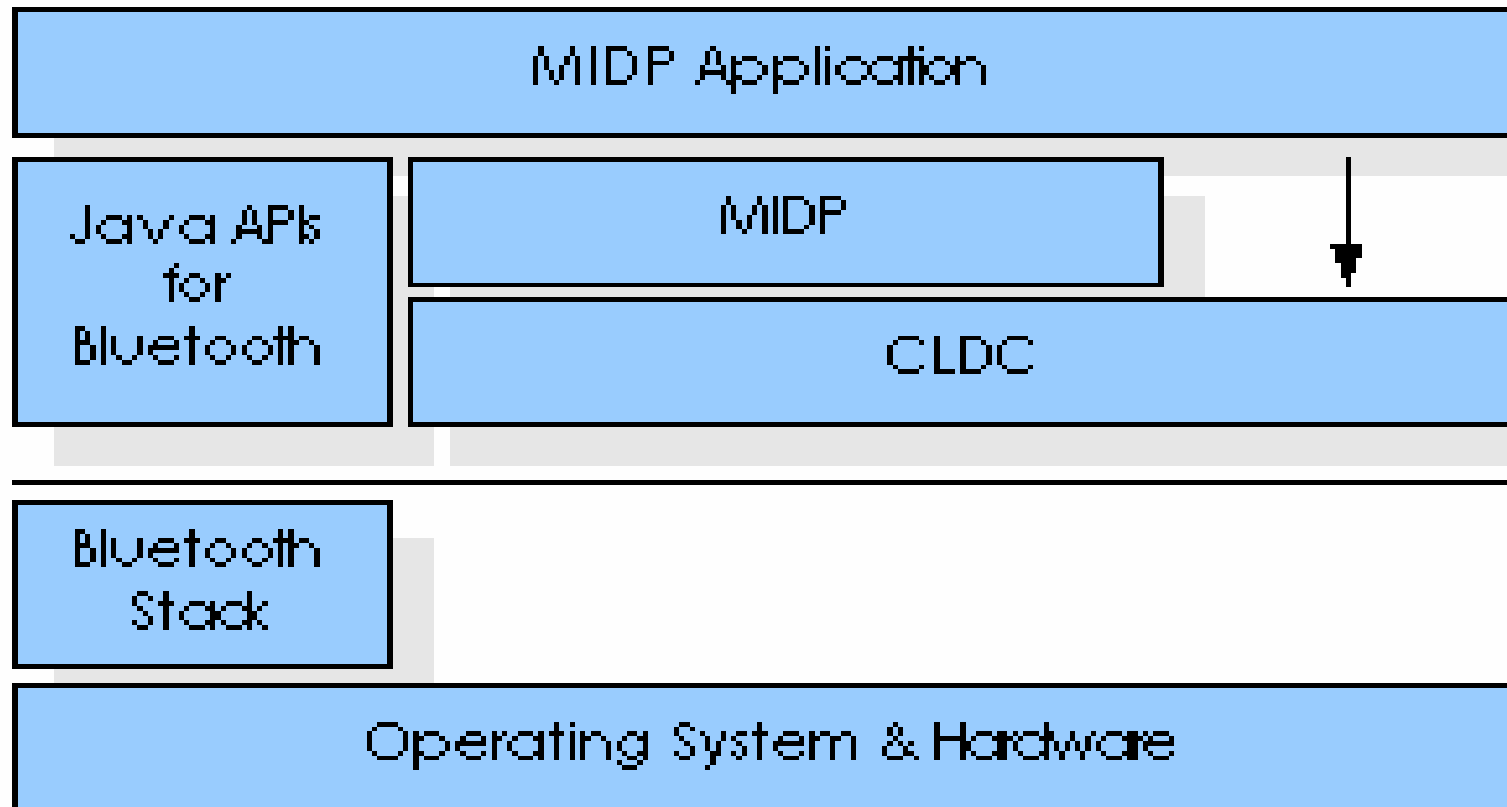
- Sviluppata dal SIG (www.bluetooth.org) si propone come una tecnologia low-cost per comunicazioni radio tra dispositivi elettronici
- Tecnologia Radio:
 - la banda di frequenza utilizzata è di 2.45 Ghz
 - velocità di 1Mb/s

CLASS	POWER RATING	RANGE
CLASS 1	100mW	100metri
CLASS 2	2.5mW	20metri
CLASS 3	1mW	10metri

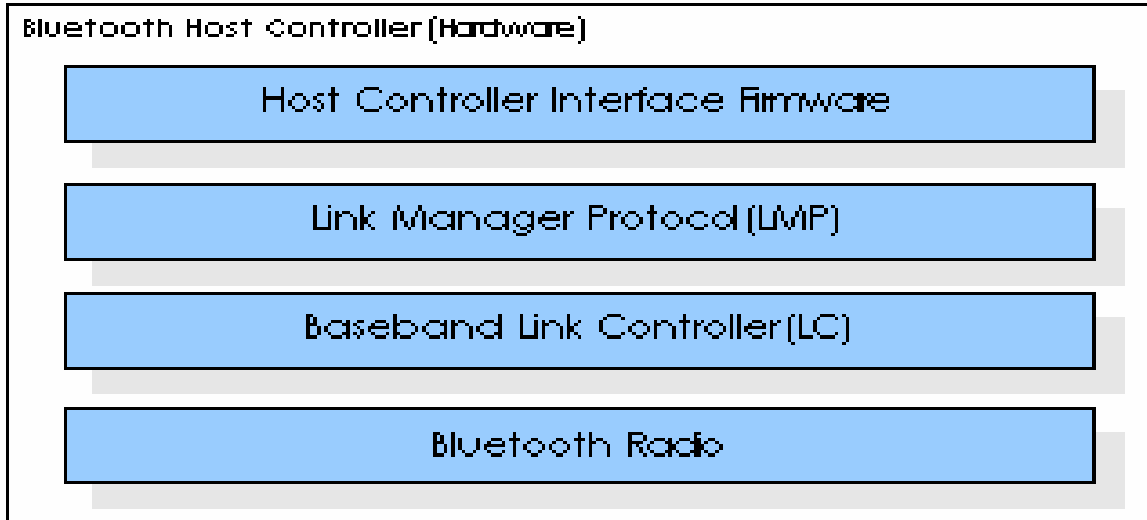
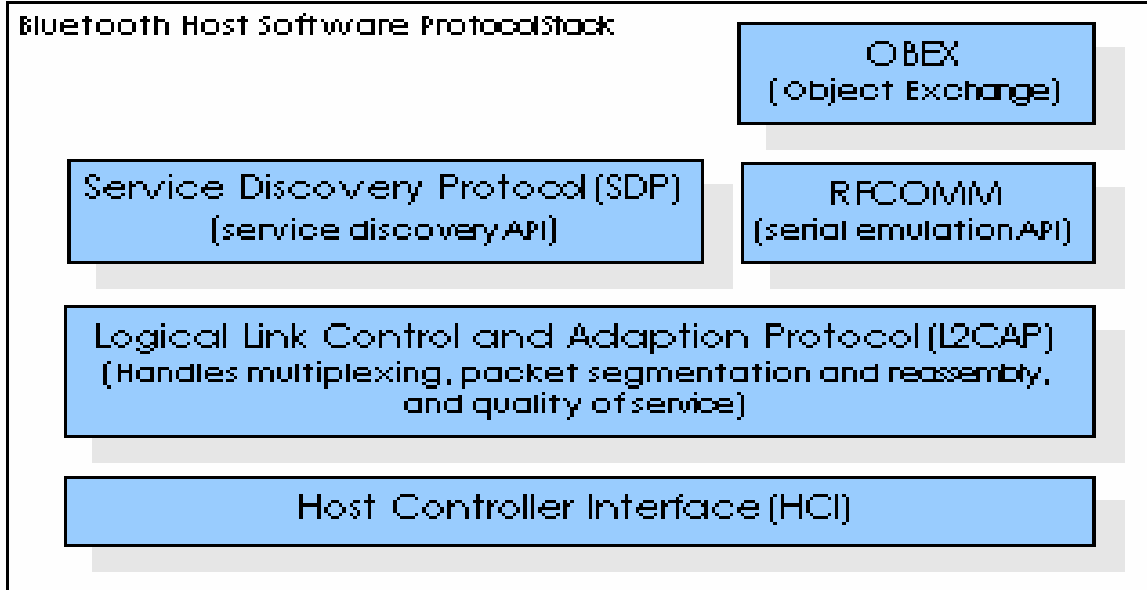
Introduzione (1)

- L'API JSR-82 per la comunicazione tramite tecnologia Bluetooth è un package opzionale non facente parte del profilo MIDP 2.0
- La presenza di tali API su un dispositivo è fortemente legata non solo alla presenza dell'hardware (antenna Bluetooth, ecc.) ma anche all'implementazione del suo “stack” e di uno strato software di basso livello per accedere ad esso

Introduzione (2)



II Bluetooth protocol stack (1)



Il Bluetooth protocol stack (2)

- Host Controller Interface (HCI)
 - lo strato di basso livello per eccellenza: si interfaccia direttamente all'hardware
- Logical Link Control and Adaptation Layer (L2CAP)
 - strato che permette la gestione dei pacchetti, relativo ri-assemblaggio e multiplexing
- (continua...)

Il Bluetooth protocol stack (3)

- (segue)
- Service Discovery Protocols (SDP)
 - strato utilizzato dalle applicazioni Bluetooth per il discovery di servizi Bluetooth disponibili
- RFCOMM
 - questo strato permette di emulare una connessione “seriale” tramite Bluetooth
- Object Exchange Protocol (OBEX)
 - tale strato permette lo “scambio di oggetti” (esempio vCard, vCalendar)

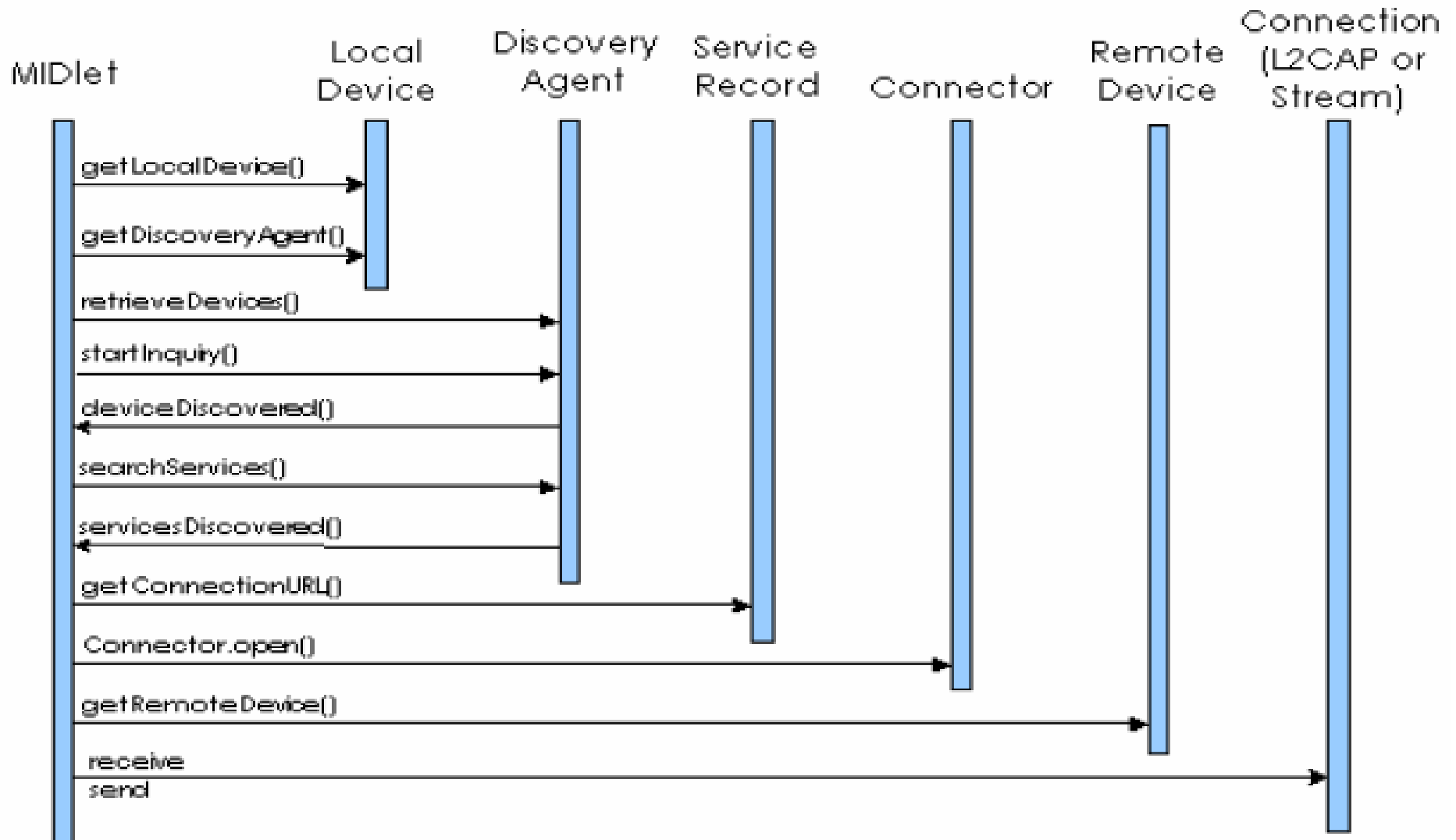
Accesso allo stack

- Tramite la JSR-82 è possibile utilizzare esplicitamente L2CAP e RFCOMM
- L2CAP
 - consigliabile per comunicazione basata su pacchetti (esempio quando si definisce un protocollo proprietario)
 - l'URL inizia per *btl2cap*
- RFCOMM
 - consigliabile per una comunicazione maggiormente semplificata basata su stream
 - l'URL inizia per *btsp*

Struttura della classica applicazione Bluetooth (1)

- Struttura lato server
 - creazione di un servizio (o utilizzo di uno esistente)
 - ascolto di eventuali richieste
 - istanziamento di una connessione
- Struttura lato client
 - ricerca di dispositivi presenti e dei relativi servizi
 - richiesta di connessione ad un servizio
 - istanziamento di una connessione
- Non è molto diversa dalle connessioni tramite socket...

Struttura della classica applicazione Bluetooth (2)



La classe LocalDevice (1)

- Contiene informazioni sul dispositivo locale utilizzato
- Definisce le funzioni base del dispositivo
- Per ottenerne un'istanza (singleton) si utilizza il metodo statico
 - `LocalDevice getLocalDevice()`: restituisce un'istanza di *LocalDevice*

La classe LocalDevice (2)

- Metodi utili
 - String *getFriendlyName()*: restituisce il nome del dispositivo locale
 - String *getBluetoothAddress()*: restituisce l'indirizzo Bluetooth del dispositivo locale
 - int *getDiscoverable()*, boolean *setDiscoverable(int mode)* :restituisce/imposta la modalità con cui questo dispositivo può essere “scoperto”
- (continua...)

La classe LocalDevice (3)

- (segue)
 - DeviceClass *getDeviceClass()*: restituisce un oggetto contenente informazioni aggiuntive sul dispositivo e sui suoi servizi (maggiori dettagli a breve)
 - DiscoveryAgent *getDiscoveryAgent()*: restituisce un oggetto capace di scoprire dispositivi e servizi (maggiori dettagli a breve)
 - ServiceRecord *getRecord(Connection notifier)*: restituisce un oggetto che descrive un servizio Bluetooth (maggiori dettagli a breve)

Il browsing dei dispositivi remoti

- Nella maggior parte delle applicazioni Bluetooth la necessità primaria è quella di determinare i dispositivi remoti presenti nel raggio di azione
- Tale Compito, spetta a:
 - *DiscoveryAgent*
 - *DiscoveryListener*

Il browsing dei dispositivi remoti la classe *DiscoveryAgent* (1)

- La classe *DiscoveryAgent* fornisce i metodi per eseguire la ricerca di dispositivi e servizi (abbiamo già visto come recuperare tale oggetto tramite la classe *LocalDevice*)
- Ci sono due modi per scoprire dispositivi, il primo è tramite il metodo
 - boolean *startInquiry*(int accessCode, *DiscoveryListener* listener): inizia la ricerca di dispositivi
 - i dispositivi trovati vengono restituiti tramite il metodo *deviceDiscovered*(...) di *DiscoveryListener* (maggiori dettagli a breve)

Il browsing dei dispositivi remoti la classe `DiscoveryAgent` (2)

- Il secondo modo è tramite il metodo
 - `RemoteDevice[] retrieveDevices(int option)`: restituisce un array di dispositivi che sono stati trovati da precedenti chiamate al metodo `startInquiry(...)` oppure che sono stati specificati come dispositivi “già noti”
- I dispositivi “già noti” sono quei dispositivi definiti nel *Bluetooth Control Center* come dispositivi spesso contattati dal *LocalDevice*
- Il metodo `retrieveDevices(...)` non esegue una ricerca, ma fornisce un metodo veloce per avere una lista di dispositivi che potrebbero essere nella zona

Il browsing dei dispositivi remoti la classe `DiscoveryAgent` (3)

- *DiscoveryAgent* permette anche la scoperta di servizi
- Ci sono due modi per cercare i servizi, se si desidera cercare un servizio su un singolo dispositivo bisogna usare il metodo
 - `int searchServices(int[] attrSet, UUID[] uuidSet, RemoteDevice btDev, DiscoveryListener discListener)`: ricerca i servizi su un dispositivo remoto avente gli UUID specificati
- Se invece non è importante quale dispositivo fornisce il servizio allora si deve usare il metodo
 - `String selectService(UUID uuid, int security, boolean master)`: cerca un servizio

Il browsing dei dispositivi remoti l'interfaccia `DiscoveryListener` (1)

- Ultimo argomento prima di passare ad un esempio e l'interfaccia *DiscoveryListener*
- *DiscoveryListener* permette di ricevere gli eventi di scoperta di dispositivi e servizi
- Fornisce quattro metodi, due per la scoperta di dispositivi e due per la scoperta di servizi

Il browsing dei dispositivi remoti l'interfaccia DiscoveryListener (2)

- I metodi da implementare sono
 - void *deviceDiscovered*(RemoteDevice btDevice, DeviceClass cod): lanciato quando viene trovato un dispositivo
 - void *inquiryCompleted*(int discType): lanciato quando si completa la ricerca di dispositivi
 - void *servicesDiscovered*(int transID, ServiceRecord[] servRecord): lanciato quando viene trovato un servizio
 - void *serviceSearchCompleted*(int transID, int respCode): lanciato quando si completa la ricerca di servizi

Il browsing dei dispositivi remoti

Esempio

BrowserSample

(No su MPowerPlayer)

La classe *RemoteDevice* (1)

- La classe *RemoteDevice* vista nel precedente esempio rappresenta un dispositivo remoto
- Tale classe fornisce le informazioni base sul dispositivo incluso l'indirizzo Bluetooth ed il suo nome
- Non è possibile istanziarla direttamente, ma la si può recuperare dalla ricerca dei dispositivi oppure utilizzando il metodo statico
 - *RemoteDevice getRemoteDevice(Connection conn)*: recupera il dispositivo connesso alla connessione *conn*

La classe RemoteDevice (2)

- Metodi utili
 - String *getFriendlyName*(boolean alwaysAsk): restituisce il nome del dispositivo
 - String *getBluetoothAddress*(): restituisce l'indirizzo del dispositivo
 - boolean *authenticate*(): tenta di autenticare il dispositivo
 - boolean *isAuthenticated*(): verifica l'autenticazione del dispositivo
 - (continua)

La classe RemoteDevice (3)

- Metodi utili (segue)
 - boolean *authorize*(Connection conn): determina il permesso di continuare ad accedere ai servizi forniti da *conn*
 - boolean *isAuthorized*(Connection conn): verifica il permesso di continuare ad accedere ai servizi forniti da *conn*
 - boolean *encrypt*(Connection conn, boolean on): attiva/disattiva l'encrptazione per questa connessione
 - boolean *isEncrypted*(): verifica l'encrptazione per questa connessione

I servizi (1)

- Mediante il Bluetooth è possibile realizzare “piccoli” server che mettono a disposizione determinati servizi
- L'UUID (Universal Unique Identifier) è un numero che identifica un determinato servizio
- Esistono degli UUID standard, per esempio *rfcomm*, *l2cap* ed *http* hanno il loro UUID predefinito (rispettivamente 0X0003, 0X0100, 0x000C)

I servizi (2)

Base UUID Value	0x0000000000000001000800000805F9B34FB	128-bit
SDP	0x0001	16-bit
RFCOMM	0x0003	16-bit
OBEX	0x0008	16-bit
HTTP	0x000C	16-bit
L2CAP	0x0100	16-bit
BNEP	0x000F	16-bit
Serial Port	0x1101	16-bit
ServiceDiscoveryServerServiceClassID	0x1000	16-bit
BrowseGroupDescriptorServiceClassID	0x1001	16-bit
PublicBrowseGroup	0x1002	16-bit
OBEX Object Push Profile	0x1105	16-bit
OBEX File Transfer Profile	0x1106	16-bit
Personal Area Networking User	0x1115	16-bit
Network Access Point	0x1116	16-bit
Group Network	0x1117	16-bit

I servizi (3)

- È possibile definire nuovi UUID per identificare nuovi servizi
- Il layer che si occupa dei servizi è SDP ed esiste un database dei servizi in ogni dispositivo Bluetooth: SDBB
- Ogni servizio deve essere registrato prima di poter essere richiesto da un client

La classe UUID (1)

- *UUID* è semplicemente una classe di comodo per definire UUID
- Per questi *UUID* è garantita l'unicità assoluta e pertanto un'istanza di questa classe è immutabile
- Le specifiche Bluetooth forniscono un algoritmo che descrive come un UUID a 16 o 32 bit dovrebbe essere convertito in un UUID a 128 bit *UUID* e pertanto questa classe fornisce un'interfaccia che permette la creazione di *UUID* a 16, 32 e 128 bit

La classe UUID (2)

- Costruttori
 - *UUID*(long uuidValue)
 - *UUID*(String uuidValue, boolean shortUUID)
- Metodi utili
 - boolean *equals*(Object value): determina se due *UUID* sono uguali

L'interfaccia *ServiceRecord* (1)

- L'interfaccia *ServiceRecord* descrive le caratteristiche di un servizio Bluetooth
- Un *ServiceRecord* contiene una serie di attributi del servizio dove ogni attributo è una copia (ID, valore)
- L'ID è un intero a 16 bit, il valore è un oggetto *DataElement*

L'interfaccia ServiceRecord (2)

Nome dell'Attributo	ID	Tipo di valore
ServiceRecordHandle	0x0000	32-bit unsigned integer
ServiceClassIDList	0x0001	DATSEQ of UUIDs
ServiceRecordState	0x0002	32-bit unsigned integer
ServiceID	0x0003	UUID
ProtocolDescriptorList	0x0004	DATSEQ of DATSEQ of UUID and optional parameters
BrowseGroupList	0x0005	DATSEQ of UUIDs
LanguageBasedAttributeIDList	0x0006	DATSEQ of DATSEQ triples
ServiceInfoTimeToLive	0x0007	32-bit unsigned integer
ServiceAvailability	0x0008	8-bit unsigned integer
BluetoothProfileDescriptorList	0x0009	DATSEQ of DATSEQ pairs
DocumentationURL	0x000A	URL
ClientExecutableURL	0x000B	URL
IconURL	0x000C	URL
VersionNumberList	0x0200	DATSEQ of 16-bit unsigned integers
ServiceDatabaseState	0x0201	32-bit unsigned integer

L'interfaccia ServiceRecord (3)

- Metodi utili
 - `int[] getAttributeIDs()`: restituisce gli ID degli attributi accessibili
 - `DataElement getAttributeValue(int attrID)`: restituisce il valore di un attributo
 - `String getConnectionURL(int requiredSecurity, boolean mustBeMaster)`: restituisce l'URL per connettersi al servizio