

Sviluppo di applicazioni J2ME

Introduzione alla piattaforma

Java 2 Micro Edition



Dipartimento di Informatica e Sistemistica
Via Claudio 21, 80125 Napoli

www.mobilab.unina.it

info@mobilab.unina.it



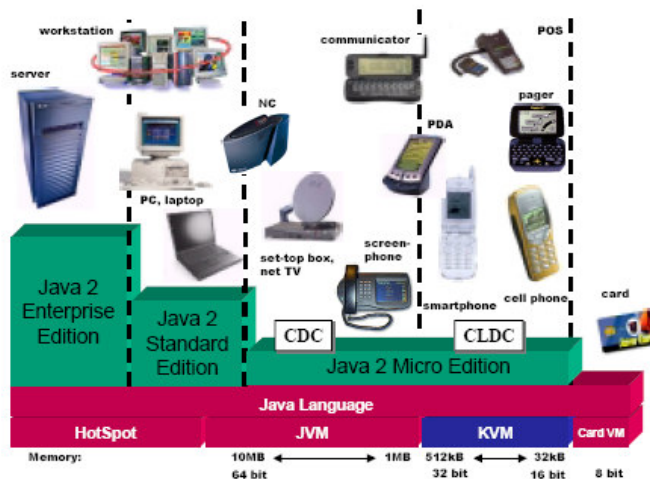
Mobile Systems

::. Contenuti della lezione

- Le diverse edizioni di Java: J2EE, J2SE, J2ME
- Architettura della piattaforma J2ME
- Le configurazioni (CDC, CLDC)
- I profili (MIDP, Personal Profile, ...)
- Java Virtual Machines per applicazioni Mobile



::. Le diverse edizioni di Java: J2EE, J2SE, J2ME



Applicazioni e Segmenti di Mercato

- **Java 2 Enterprise Edition (J2EE)**

Per imprese che hanno bisogno di fornire ai propri clienti e/o dipendenti soluzioni server-side solide e scalabili.

- **Java 2 Standard Edition (J2SE)**

Per il vasto mercato delle applicazioni desktop (Es.: dai videogiochi agli ambienti di sviluppo)

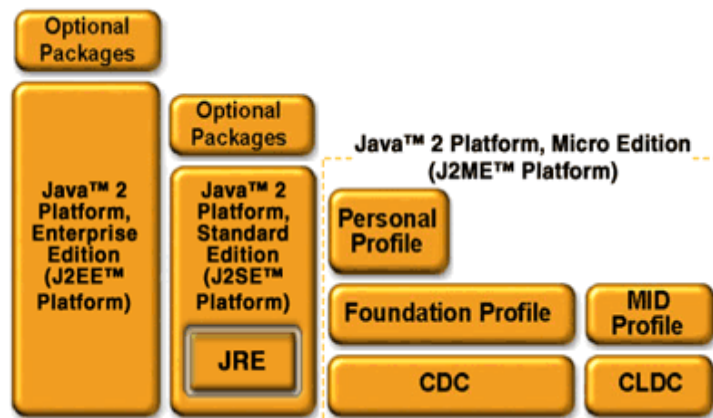
- **Java 2 Micro Edition (J2ME)**

Per i bisogni di:

- produttori di dispositivi che fabbricano una notevole varietà di dispositivi con caratteristiche hardware distinte;
- Fornitori di servizi che intendono distribuire contenuti su dispositivi mobili;
- Creatori di contenuti che intendono creare interfacce attraenti per dispositivi mobili



:: Le diverse edizioni di Java: J2EE, J2SE, J2ME



Applicazioni e Segmenti di Mercato

- **Java 2 Standard Edition (J2SE)**

- Edizione "standard" della piattaforma Java.
- Utilizza Virtual Machines per ambienti desktop, solitamente in configurazione *Client*

- **Java 2 Enterprise Edition (J2EE)**

- Include librerie aggiuntive per la realizzazione di applicazioni server di tipo *Enterprise*
- Utilizza virtual machines per ambienti desktop, solitamente in configurazione *Server*

- **Java 2 Micro Edition (J2ME)**

Architetturalmente più complessa:

- Il set di librerie a disposizione è dato dalla combinazione di un *profilo* e di una *configurazione*
- Le Virtual Machines utilizzate possono variare fortemente a seconda della tipologia di dispositivo utilizzato (Es.: smartphone vs. PDA)



::. Profili e Configurazioni

Challenges for java on small devices

- *Computing capability: CPU and memory*
- *Power*
- *Connectivity*
- *User interface and form factors*

J2ME design goals

- *Portability*
- *Rich API*
- *Network support*
- *Security model*

Solutions for a common GUI

Abstracting network and persistence support

Kilobyte Virtual Machine (KVM) – 1999 JavaOne Conference



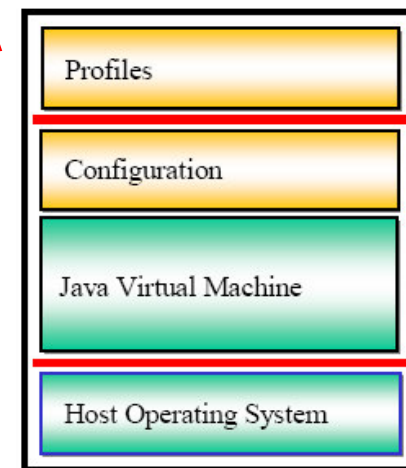
::: Profili e Configurazioni

PROFILE

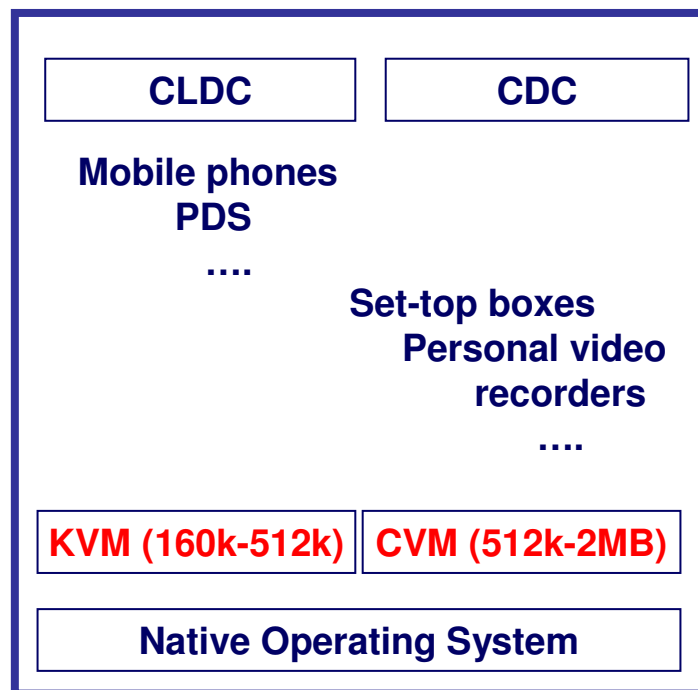
- Direttamente visibile agli utenti
- Minimo Set di APIs disponibile su una determinata famiglia di dispositivi
- Definiti a partire dalla CONFIGURATION
- Per ogni CONFIGURATION possono esistere più PROFILES

CONFIGURATION

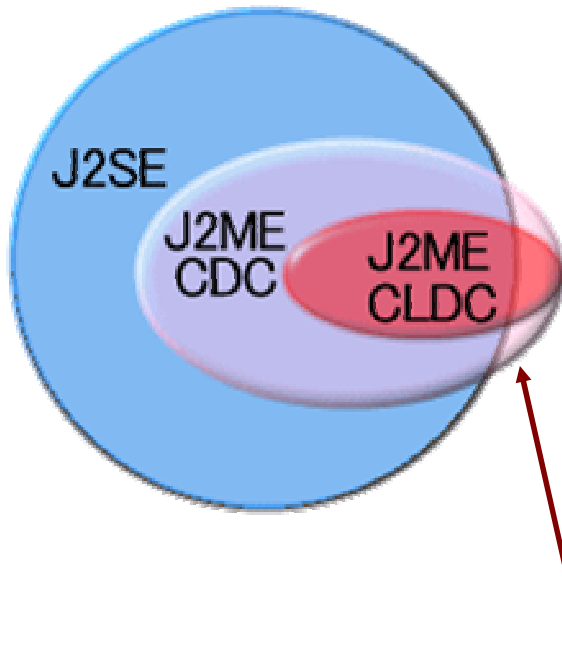
- Poco visibile agli utenti
- Minimo Comune Denominatore per una vasta classe di dispositivi
- Definisce il minimo set di caratteristiche della JVM
- Definisce il minimo set di librerie da utilizzare



::. Profili e Configurazioni



::: CDC vs. CLDC



- Le due configurazioni non possono essere utilizzate insieme
- La maggior parte delle funzionalità sono state ereditate da J2SE
- CLDC è un sottoinsieme di CDC
- CDC e CLDC non sono dei sottoinsiemi di J2SE

**Librerie specifiche per dispositivi mobili, in particolare il
*Generic Connection Framework (GCF)***



::. Configurazione CLDC (JSR – 139)

***Progettato per dispositivi fortemente limitati
dal punto di vista della memoria***

**Ambiente Java “ridotto” che lavora efficientemente con un
memory footprint compreso tra 128 e 256 KBytes.**

- La Virtual Machine ha caratteristiche limitate
(Es.: non supporta l'aritmetica Floating Point)
- Supporta esclusivamente un piccolo sottoinsieme delle librerie “core”
- Fornisce in compenso nuove librerie per la gestione della connettività
(Es.: via Bluetooth)

Il profilo CLDC risulta particolarmente idoneo per *smartphones*

CLDC = Connected Limited Device Configuration



::. Configurazione CLDC (JSR – 139)

CLDC e JVMSpec (Java Virtual Machine Specification)

Caratteristiche fortemente limitate:

- **NO** floating point →
- **NO** JNI (Java Native Interface) →
- **NO** class loaders definiti dall'utente →
- **NO** reflection →
- **NO** daemon threads →
- **NO** finalizzazione →
- **NO** "weak" references →
- **LIMITED** error handling →

(molte classi derivate da Java.lang.error non esistono)

**Mancanza
Supporto Hardware**

Problemi di sicurezza

**Vincoli
Di
Memoria**



::. Configurazione CLDC (JSR – 139)

CLDC e JLS (Java Language Specification)

Non compatibile al 100% :

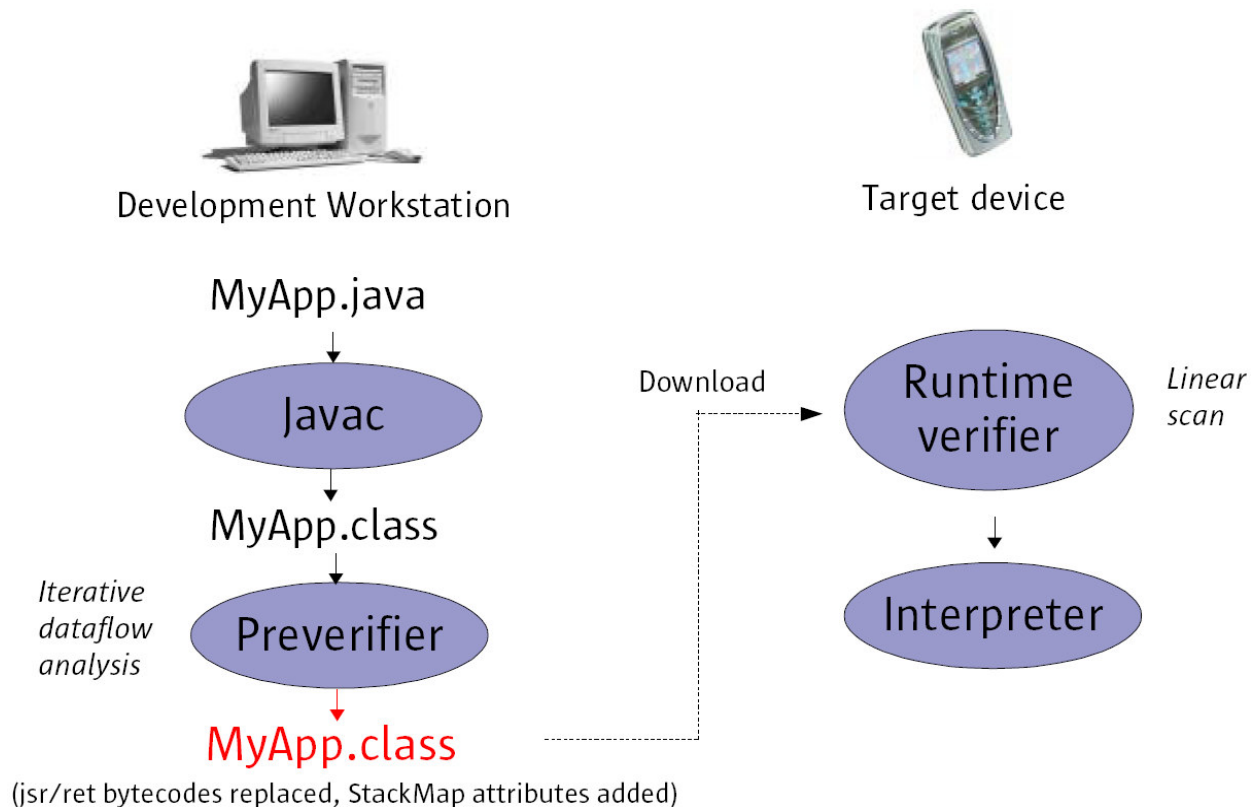
- **NO** supporto per tipi a virgola mobili
- **NO** supporto la serializzazione (es. manca RMI)
- **LIMITED** Networking e I/O

Ad eccezione di queste differenze, le implementazioni della JVM per configurazione CLDC devono essere completamente conformi alla specifica del linguaggio Java



:: Configurazione CLDC (JSR – 139)

Sicurezza: verifica Class Files



Package disponibili

- *Java.io*
- *Java.lang*
- *Java.util*
- *Javax.microedition.io*



::. Configurazione CDC (JSR – 218)

***Progettato per ottenere la massima compatibilità
con J2SE su dispositivi resource-constrained***

**Ambiente Java flessibile che lavora efficientemente con soli 2 MB
di RAM e 2MB di ROM.**

- Supporta completamente tutte le specifiche della Java Virtual Machine;
- Include tutte le librerie "core" (come `java.lang`);
- supporta tutte le librerie della stander edition, scalate ed ottimizzate per adattarele a dispositivi resource constrained
 - *Alcune Interfacce Modificate*
 - *Alcune Classi Rimosse*

Dispositivi "indicati" per il profilo CDC

PDA's, set-top boxes, stampanti di rete, routers, residential gateways,
alcuni telefoni mobili e VoIP

CDC = Connected Device Configuration



:: Configurazione CDC (JSR – 218)

Interfaccia Grafica (GUI)

Scenario	Descrizione	Esempio
No GUI	Dispositivo senza alcuna interfaccia grafica per l'utente	Stampante di rete
Proprietary GUI	Semplice GUI proprietaria (es.: testo e bottoni)	<i>Multifunzione</i>
Raw graphics device	Necessita di un GUI toolkit portabile (QT, GTK)	<i>Set-Top Box</i>
Native GUI toolkit	GUI Toolkit disponibile nel dispositivo, utilizzato da librerie come AWT	<i>Telefoni cellulari, PDAs</i>
Applet support for Web Browsing	Il dispositivo può navigare sul web e richiede supporto per le applet	<i>Telefoni cellulari, PDAs</i>
Rich GUI toolkit	Il dispositivo richiede sofisticate caratteristiche grafiche. Sono richiesti packages addizionali (Es.: AGUI)	<i>Dispositivi embedded sofisticati</i>



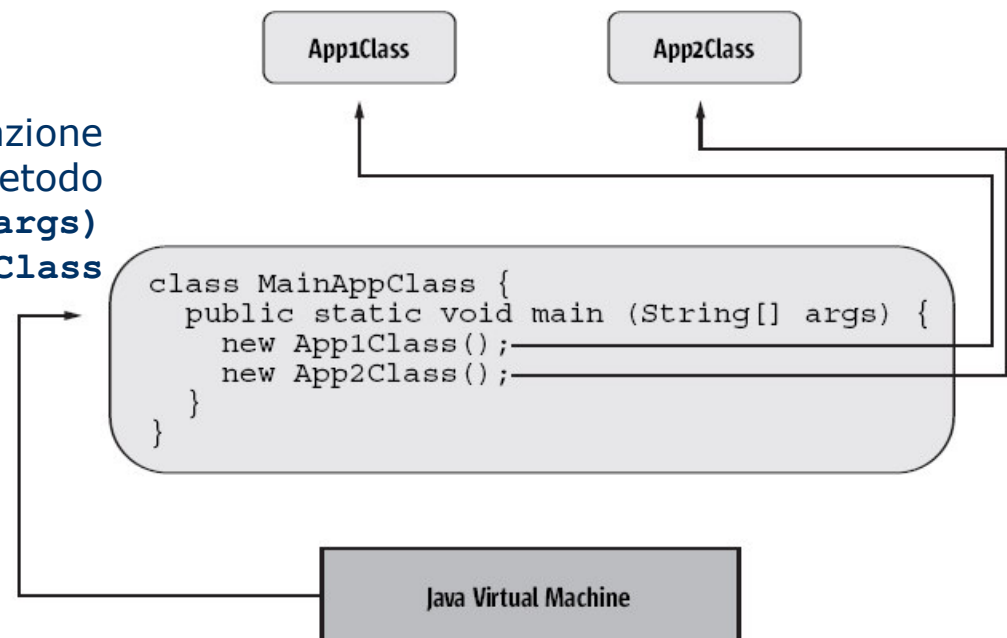
:: Configurazione CDC (JSR – 218)

Modello di programmazione: Applicazioni STANDALONE

Classica Applicazione Java

2. L'esecuzione dell'applicazione
inizia dal metodo
`public static void main(String[] args)`
di `MainAppClass`

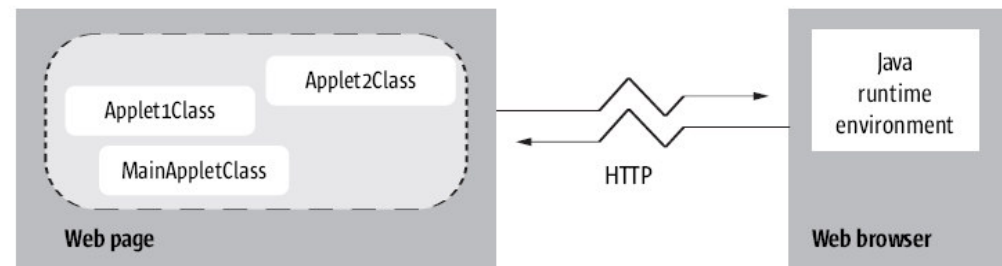
1. `java MainAppClass`



::. Configurazione CDC (JSR – 218)

Modello di programmazione: **APPLETS**

Classico Utilizzo di Java incorporato in un Web Browser



1. Il Browser dal Web scarica una pagina con una applet (descritta da 3 classfiles)
2. Il browser effettua il rendering della pagina HTML e passa alla JVM i classfiles
3. La JVM esegue MainAppletClass, che deve contenere tutti i metodi per la gestione del ciclo di vita di una applet

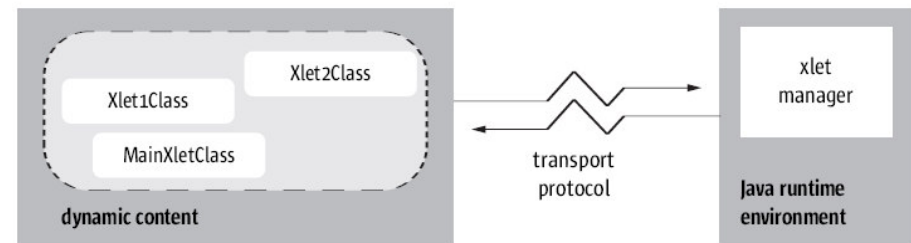
NB: Si tratta sempre di applicazioni GUI-based (basate su AWT)



::: Configurazione CDC (JSR – 218)

Modello di programmazione: XLETS

Simili alle applet nell'intento, ma
differenti nel design



- GUI-Independent (Non è necessario che siano applicazione grafiche, né che siano realizzate con AWT)
 - *Possono essere utilizzate su un qualsiasi tipo di dispositivo CDC-enabled*
- Gestite da un apposito xlet Manager
- Le xlet comunicano tra di loro via RMI (Remoted Method Invocation)



::: I profili: Foundation Profile (JSR – 219)

- Per configurazione CDC
- Profilo “di base”
- Fornisce un set di classi minimale (supporto di rete ed I/O)
- Non include supporto per grafica od altri tipi di GUI

Packages supportati

`java.io, java.lang, java.lang.ref, java.lang.reflect, java.math, java.net,
java.security, java.security.acl, java.security.cert, java.security.interfaces,
java.security.spec, java.text, java.util, java.util.jar, java.util.zip,
javax.microedition.io`

Unico package non J2SE
(Generic Connection Framework)



::: I profili: Personal Basis Profile (JSR – 217)

- Per configurazione CDC
- Profilo per lo sviluppo di applicazioni “leggere”
- Estende il set di classi fornito con il Foundation Profile
- Include supporto (*limitato*) per AWT
- Fornisce supporto a runtime per JavaBeans ed xlets

Packages Supportati

`java.awt, java.awt.color, java.awt.event, java.awt.image`

`java.beans`

`java.rmi, java.rmi.registry`

`javax.microedition, javax.microedition.xlet.ixc`

`java.applet`

... e tutti i packages del Foundation Profile

Solo supporto
a tempo di
esecuzione

Supporto minimale
esclusivamente per
xlets



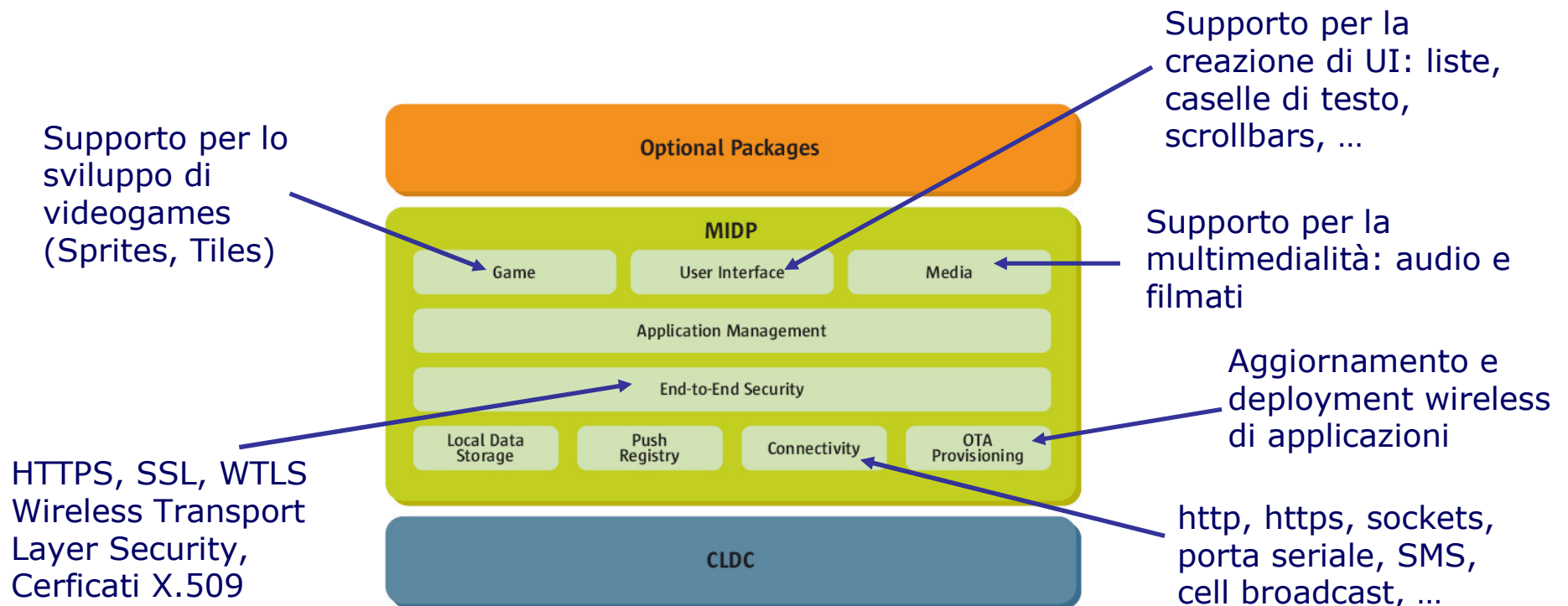
::. I profili: Personal Profile (JSR – 216)

- Per configurazione CDC
- Profilo per lo sviluppo di applicazioni “complete”
- Estende il set di classi fornito con il Personal Profile
- Include supporto *completo* per AWT
- Punto di migrazione per tecnologie Java “legacy” come PersonalJava



::: I profili: Mobile Information Device (JSR – 118)

MIDP per configurazione CLDC



::: Virtual Machines: Insigna Jeode

- Supporta configurazioni CDC e CDLC (CDC 1.0.1, CLDC 1.1)
- Disponibile per Windows, Linux/x86, Windows CE, Linux/ARM, PalmOS, VxWorks, ...
- Licenza Commerciale
- Supporto per *Adaptive Dynamic Compilation*
(*Individua a tempo di esecuzione i performance bottlenecks e li ottimizza*)
- Garbage Collection Concorrente (single-threaded)
(*fase 1: collection – fase2: compattazione*)
- Mapping 1-1 tra threads Java e threads del Sistema Operativo



::: Virtual Machines: IBM J9

- Supporta configurazioni CDC e CDLC (CDC 1.0.1, CLDC 1.1)
- Disponibile per Windows, Linux/x86, Windows CE, Linux/ARM, PalmOS, VxWorks, ...
- Licenza commerciale
- Affianca alla Just-In-Time compilation (JIT) la Ahead-Of-Time Compilation (AOT)
- Multithreading con mapping 1-1 in configurazione CDC
- Multithreading gestito dalla JVM in configurazione CLDC
(mapping N-1 su threads di livello S.O.)
- Garbage Collection "generazionale"
(area heap divisa in più generazioni sulle quali agiscono diversi collectors)
- *WebSphere Application Server per lo sviluppo e il debuggging di applicazioni*



::: Virtual Machines: Sun CDC Hotspot Implementation

- Supporta configurazione CDC
- Disponibile per Windows CE, Linux/x86, Linux/ARM (e tante altre)
- Licenza CPL (codice sorgente liberamente disponibile)
- Compilazione JIT
- Garbage Collection Generazionale (seriale)
- Mapping 1-1 tra threads Java e threads del Sistema Operativo
- Supporto per Java Native Interface (JNI), Java Virtual Machine Debugging Interface (JVMDI), Java Virtual Machine Profiling Interface (JVMPI)

Può essere rivista come un adattamento a sistemi resource constrained della JVM utilizzata nei sistemi desktop.
(La CDC HI discende direttamente dalla JVM 1.3)



::: Virtual Machines: Sun Kilo Virtual Machine (KVM)

- Virtual Machine di riferimento per CLDC versione 1.0
- Disponibile per Linux, PalmOS, Solaris, Symbian (via porting Nokia)
- Licenza CPL (codice sorgente liberamente disponibile)
- Compilazione JIT e AOT
- Garbage Collection Generazionale (seriale)
- Multithreading gestito dalla JVM
- Dimensioni della Virtual Machine estremamente ridotte: 50 – 80 KB
- Footprint ridotto: Le applicazioni Java posso girare anche su dispositivi con soli 128 KB di RAM
- JNI non supportata



::: Virtual Machines: Sun CLDC Hotspot Implementation

- Virtual Machine di riferimento per CLDC versione 1.1
- Disponibile per Linux(Arm ed x86), Symbian (via porting Nokia)
- Licenza CPL (codice sorgente liberamente disponibile)
- Compilazione JIT "suspendable", con individuazione Hotspots
- Supporto per AOT compilation
- Garbage Collection Generazionale (seriale)
- Multithreading gestito dalla JVM
- Esecuzione In-Place (trasformazione di classfiles in eseguibili)
- La JVM dispone di molte opzioni di configurazione, consentendo quindi un ottimo tuning delle performances sulla grande varietà di dispositivi su cui può essere installata



::. Virtual Machines: Sun CLDC Hotspot Implementation

MultiTasking

Supporto a livello VM per l'esecuzione concorrente di più Midlets

- Scheduling "equo" e preemptive delle diverse midlet in esecuzione
- Lo scheduling viene gestito a livello VM (non si fa affidamento sulle caratteristiche di Multitasking del S.O. sottostante)
- Isolamento delle Midlets
- Recupero delle risorse occupate dalle midlet una volta che queste terminano



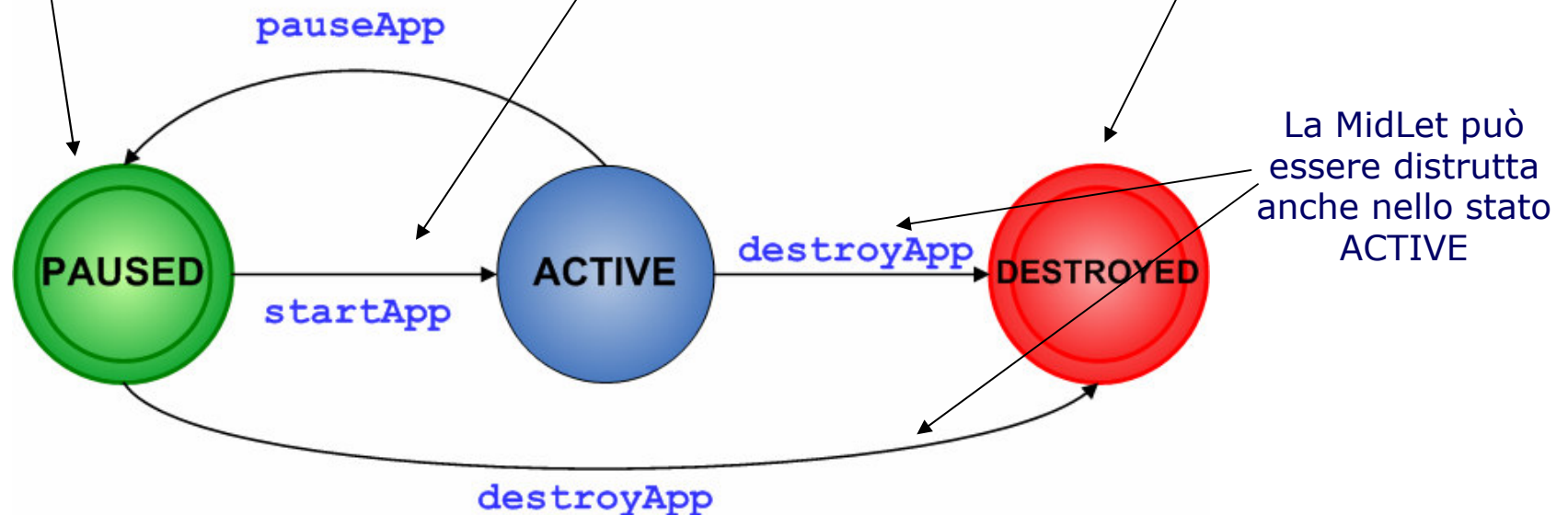
::. Application Models: MidLet

Ciclo di Vita

La MidLet è stata costruita ed è attualmente inattiva (si noti l'assenza dello stato loaded)

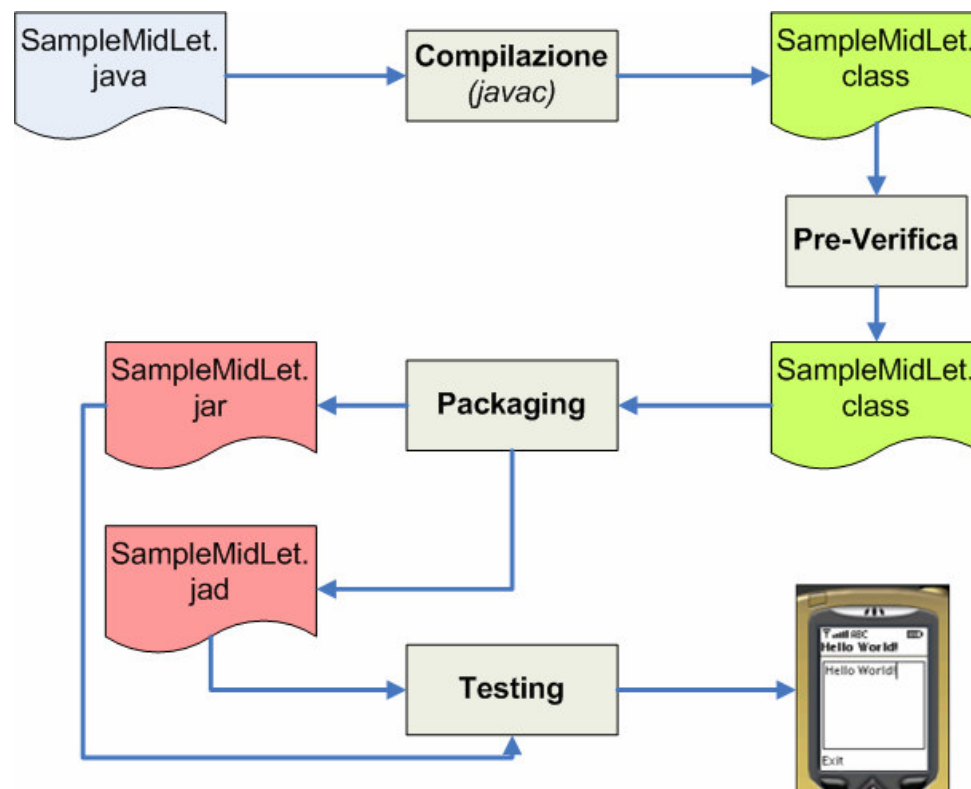
startApp provoca l'attivazione della Midlet. Se questa non può avvenire, si solleva un'eccezione e la MidLet viene distrutta

La MidLet è terminata e pronta per essere "riciclata" dal Garbage Collector



::: Ciclo di sviluppo di una MidLet

Il ciclo di sviluppo di una Applet o di una XLET è identico a quello di una tradizionale applicazione Java



1. **Compilazione** – bisogna fornire il classpath delle librerie J2ME a javac
2. **Pre-Verifica** – altera il classfile al fine di alleggerire il meccanismo di verification a runtime (come richiesto dalla configurazione CLDC)
3. **Packaging** – produce la MidLet suite, costituita da un jar includente tutti i class file e da un application descriptor, il JAD
4. **Testing** – collaudo della MidLet su di un emulatore (generico o specifico per una determinata classe di dispositivi)



::: Java Application Descriptor (JAD)

- Contiene le informazioni sull'applicazione, mostrate all'utente prima che si proceda con il download e l'installazione del software
- Il JAD file fornisce all'application manager maggiori informazioni sul contenuto del JAR file, per decidere se la MIDlet suite può essere implementata sul dispositivo (configurazione, profilo...) e se il bytecode proviene da un dominio autorizzato ad utilizzare certe funzionalità
- Può essere un modo per passare parametri a una MIDlet senza modificare il JAR file (un web service, ad esempio, potrebbe inserire dinamicamente delle informazioni nel JAD)
- Il JAD file è simile al manifest file come coppie attributo-valore
- Cinque attributi sono obbligatori:
 - MIDlet-Name
 - MIDlet-Version
 - MIDlet-Vendor
 - MIDlet-*n*
 - MIDlet-Jar-URL



::: Java Application Descriptor (JAD)

Attributo	Descrizione
MIDlet-Name	Nome della MIDlet suite
MIDlet-Version	Numero di versione della MIDlet
MIDlet-Vendor	Nome del venditore della MIDlet
MIDlet-n	Attributo per la MIDlet. I valori sono: nome della MIDlet, icona opzionale, e nome della classe MIDlet
MIDlet-Jar-URL	Locazione del JAR file
MIDlet-Jar-Size	(Opzionale) Dimensione del JAR file in bytes
MIDlet-Data-Size	(Opzionale) Dimensione minima (in byte) per la memorizzazione dei dati persistenti
MIDlet-Description	(Opzionale) Descrizione della MIDlet
MIDlet-Delete-Confirm	(Opzionale) Richiesta di conferma di rimozione della MIDlet suite
MIDlet-Install-Notify	(Opzionale) Invia lo stato dell'installazione a un dato URL

```
MIDlet-Version: 1.0.2
MIDlet-Vendor: Totore Consulting
MIDlet-Jar-URL:
http://www.mobilab.unina.it/hello.jar
MicroEdition-Configuration: CLDC-1.1
MicroEdition-Profile: MIDP-2.0
MIDlet-1: Demo1, /image/img1.png ,
org.mobilab.midlets.HelloWorld
MIDlet-Jar-Size: 10819
MIDlet-Name: HelloWorld
```



::: Sviluppo MIDP: Elementi Base

- Ogni MidLet deve ereditare dalla classe `javax.microedition.midlet.MIDlet`
- I metodi di tale classe consentono all'AMS di creare, avviare, mettere in pausa e distruggere una MidLet.
- Una MidLet deve pertanto implementare almeno i seguenti metodi:
 - `void startApp()`
 - `void pauseApp()`
 - `void destroyApp(boolean unconditional)`



:: Application Models: MidLet

- Managed Application:
viene gestita un software special purpose (AMS – Application Management System)
- L'AMS deve essere incorporato nel dispositivo (cellulari, smartphones o pagers): le midlet potrebbero essere interrotte da eventi esterni (come una telefonata)

```
import javax.microedition.midlet.*;

public class BasicMIDlet extends MIDlet
{
    public BasicMIDlet() {
        // constructor - don't do much here
    }

    protected void destroyApp(boolean unconditional)
        throws MIDletStateChangeException {
        // the system destroys the MIDlet
    }

    protected void pauseApp() {
        // the system pauses the MIDlet
    }

    protected void startApp()
        throws MIDletStateChangeException {
        // the system activates the MIDlet
    }
}
```



::. Application Models: MidLet

Contesto Operativo

- Ottenere proprietà' di inizializzazione: **getProperty**
Le proprietà di inizializzazione sono contenute in un apposito descrittore chiamato Java Application Descriptor
- Richiedere la riattivazione della MidLet: **resumeRequest**
La richiesta viene inoltrata all'AMS, che in seguito deciderà se e quando riattivare la MidLet (re)invocando il metodo startApp
- Mettere in pausa la MidLet: **notifyPaused**
- Distruggere la MidLet: **notifyDestroyed**



:: NetBeans

- Vendor: **Sun Microsystems**
- Licenza: **CDDL** (Common Development and Distribution License)
Licenza world-wide, royalty-free, non exclusive. Variazioni al codice sorgente consentite
- Disponibilità: **Windows / Linux / Solaris / MacOS**
- Ultima versione conosciuta: **6**
- J2ME viene supportata mediante Add-Ons:
 - Configurazione CLDC: **NetBeans Mobility Pack for CLDC (ver. 7.3)**
 - Configurazione CDC: **NetBeans Mobility Pack for CDC (ver. 2.0)**
- Virtual Machine di riferimento: **Sun CLDC Hotspot Implementation (CLDC), Sun CDC Hotspot Implementation (CDC)**
- URL per il download: www.netbeans.org



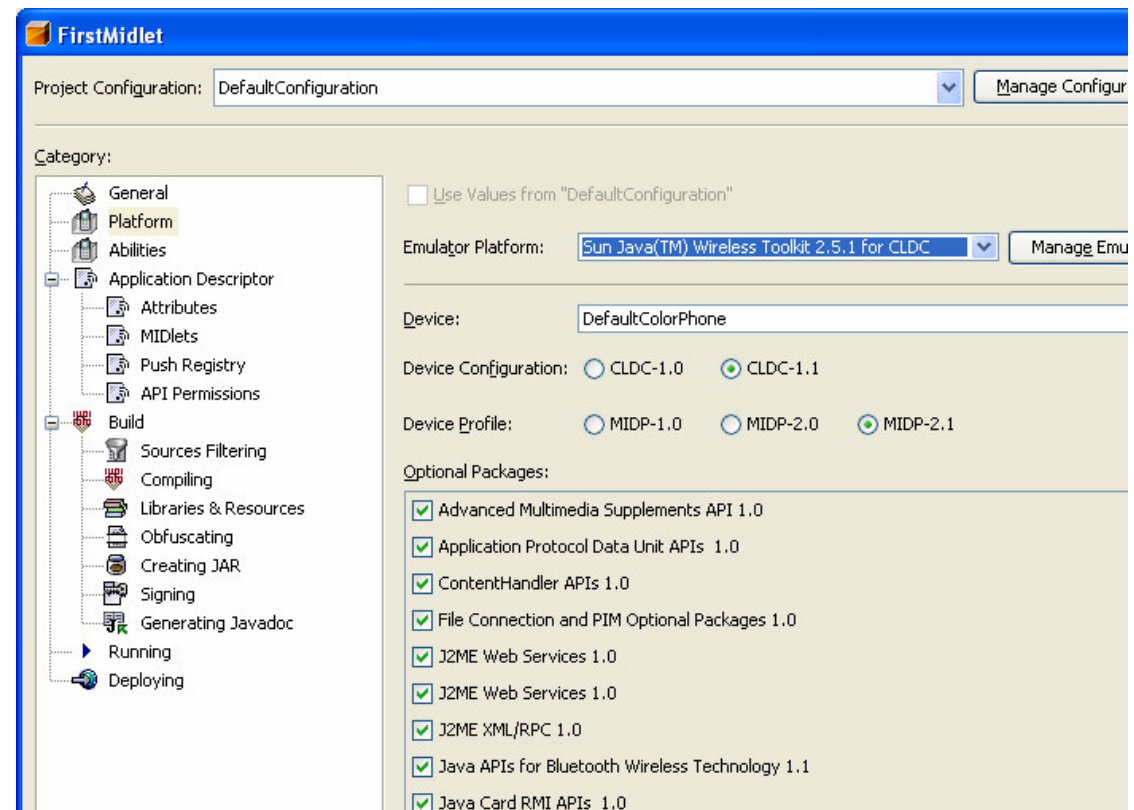
:: NetBeans

Caratteristiche Supportate (*Optional Packages*)

I packages Opzionali disponibili dipendono dalla piattaforma utilizzata per lo sviluppo di un singolo progetto (gestibile dalle proprietà del progetto).

Ad esempio, utilizzando il Wireless Toolkit 2.5.1 come piattaforma si hanno a disposizione tutte le API precedentemente elencate

Utilizzando invece una piattaforma proprietaria (Es.: Nokia S40), si hanno a disposizione Vendor-Specific APIs, come "Nokia User Interface"



:: NetBeans – Screen Designer

Preview Form

la dimensione dello schermo dipende dal profilo di emulazione selezionato

Riepilogo comandi assegnati

The screenshot displays the NetBeans IDE interface for the Screen Designer. The main window shows a preview of a form titled "helloForm [Form]". The form contains a text input field with the label "Qual'è il tuo nome?", a radio button group for "Di che sesso sei?" (Uomo, Donna, Gay/Lesbica), and a progress bar for "Qual'è il tuo livello di apprendimento?". To the right of the preview, the "Assigned Commands" section lists "exitCommand" (exit MIDlet) and "okCommand1" (display alert1 [Alert]). Below this, the "Assigned Item Commands" section shows "No command assigned". On the far right, the "Palette" window is open, showing various UI components categorized under "Commands", "Form Items", "Elements", and "Resources". The "Form Items" category is selected, showing components like ChoiceGroup, Gauge, Spacer, TextField, DateField, ImageItem, StringItem, and TableItem. Below the palette, the "helloForm [Form] - Properties" window is open, showing properties for the form, including Title, Ticker, Code Properties, Instance Name (helloForm), and Lazy Initialized (checked).

**Palette
elementi
LCD UI**

**Finestra
proprietà
elemento
corrente**



:: NetBeans – Flow Designer

Consente di specificare il flusso applicativo tra diversi Displayables ed element dei forms, associando azioni alla selezione di comandi

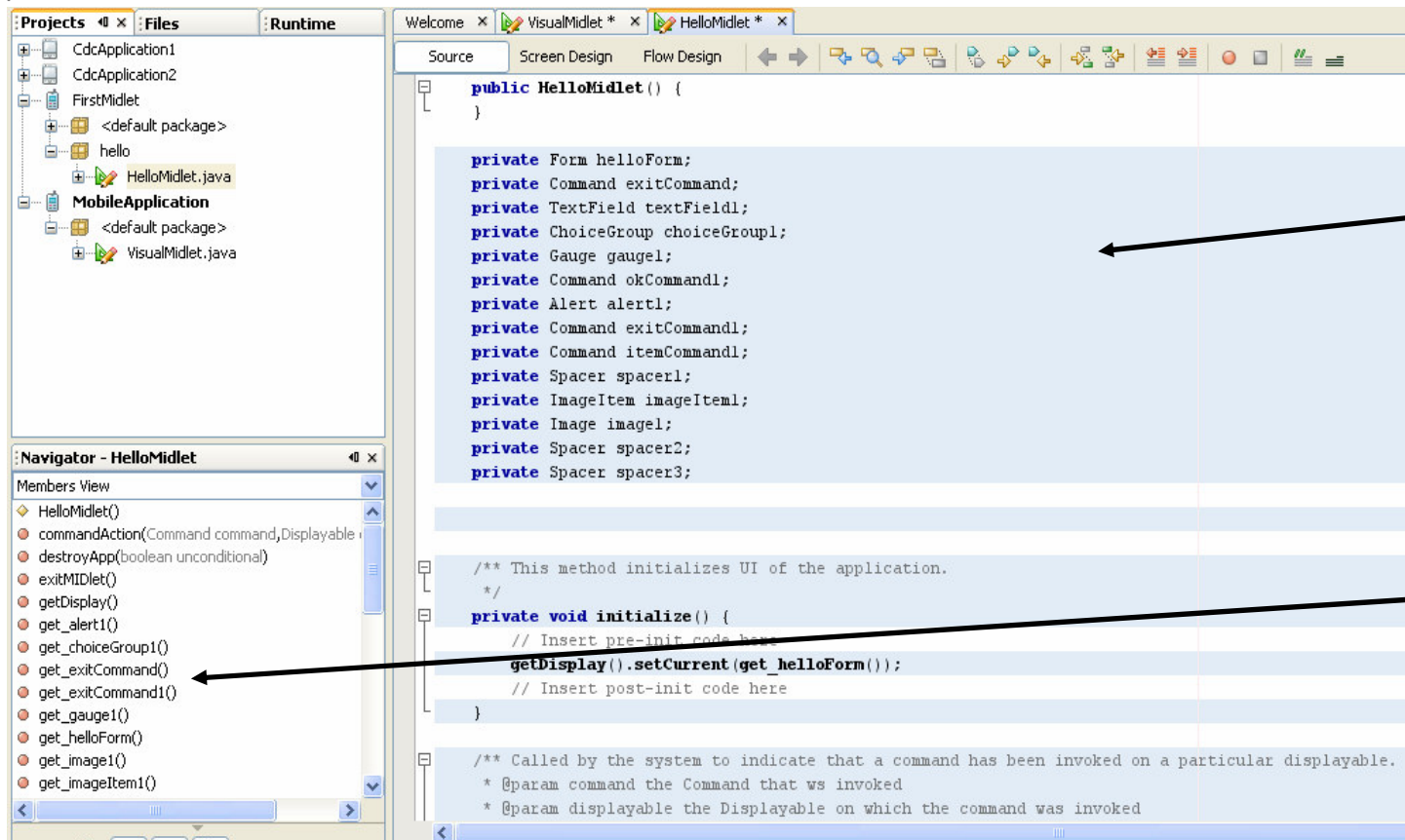
The screenshot displays the NetBeans IDE with the Flow Designer tab active. The main workspace shows a flow diagram for a MIDP-2.0 application. It starts with a 'Mobile Device' icon, followed by a 'Start Point' leading to a 'helloForm [Form]' component. From the 'helloForm', there are two paths: one labeled 'Ok' leading to an 'alert1 [Alert]' component, and another labeled 'Exit' leading to an 'Exit' action. The 'alert1 [Alert]' component has two outgoing paths: 'Dismiss' leading back to the 'Start Point' and 'Exit' leading to another 'Exit' action. The 'Action' dialog is open on the right, showing the 'Switch to screen' option selected. The 'Target screen' is set to 'alert1 [Alert]' and the 'Alert Forward screen' is set to 'helloForm [Form]'. The 'Generated code' field contains the following code:

```
getDisplay().setCurrent(get_alert1(), get_helloForm());
```

The 'MIDP Design: HelloMidlet - Properties' window is also visible at the bottom right, showing the 'Version' as 'MIDP-2.0' and the 'Start Point Action' as 'Switch Screen'.



:: NetBeans – Source Editor



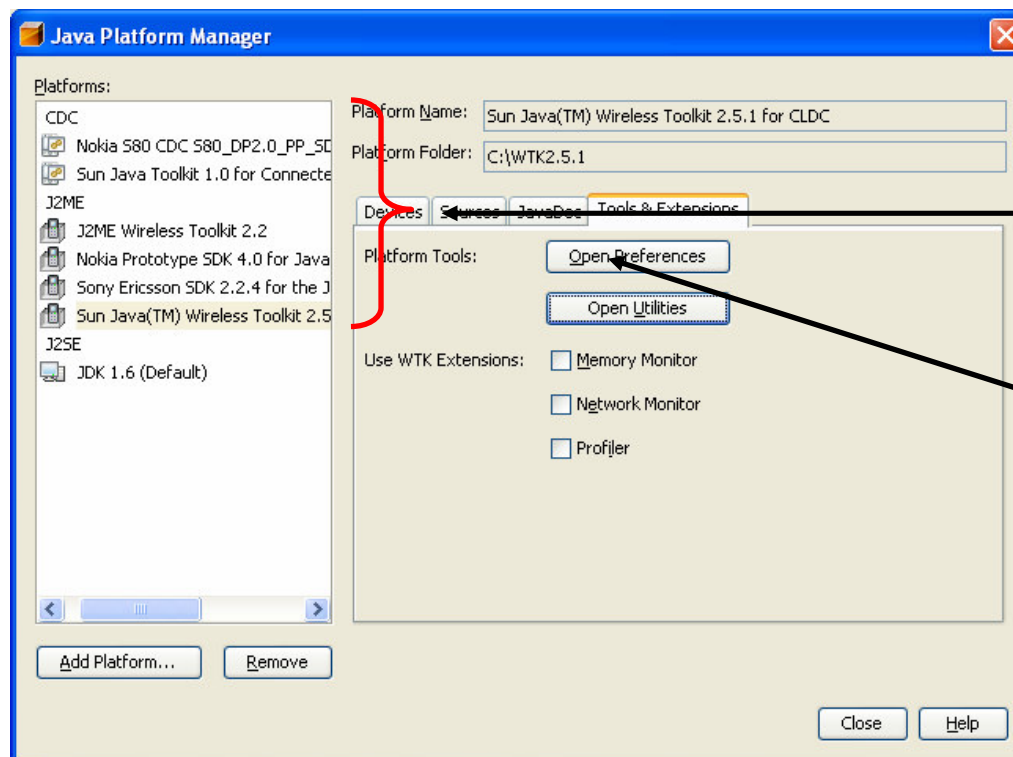
Codice Auto-Generato (non modificabile dal programmatore)

Elenco metodi classe corrente



:: NetBeans – Configurazione Emulatore

Consente di configurare l'emulatore in cui far girare la Midlet (o l'applicazione CDC).
Dal Menù tools selezionare Java Platform manager

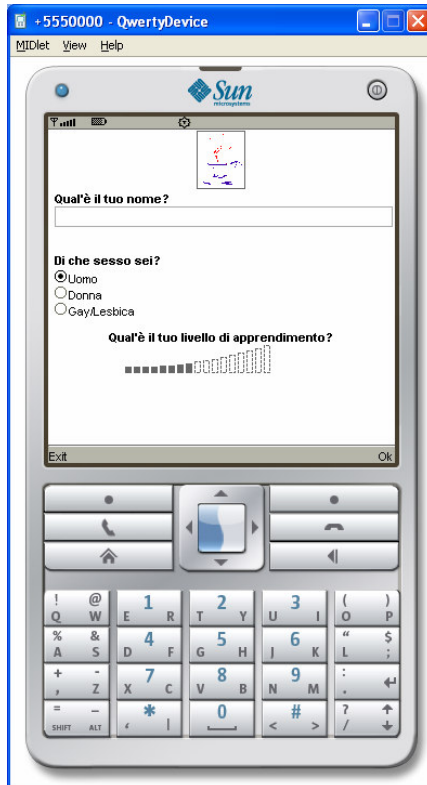
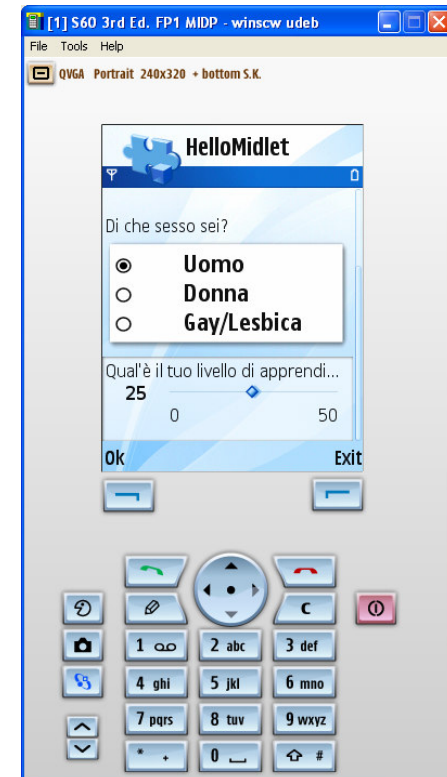


Piattaforme CDC / CLDC

Accesso all'utility di
configurazione
dell'emulatore
(software esterno)



::: NetBeans – Esecuzione

**Qwerty device***J2ME Wireless Toolkit 2.5.1***Default Color Phone***J2ME Wireless Toolkit 2.2***S60 Emulator***Nokia S60 3° Edition SDK for MIDP*

::. Applicazioni di rete con J2ME

- Generic Connection Framework (GCF)
- Java APIs for Bluetooth Wireless Technology (JSR-82)

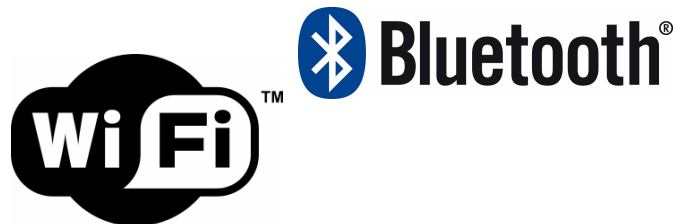


::. Generic Connection Framework (GCF)

Connettività

Non solo chiamate vocali, ma anche (e forse oggi soprattutto):

- SMS ed MMS
- Bluetooth
- Wi-Fi
- UMTS
- con diversi protocolli di trasporto (TCP, UDP, RTP, ...) ...
- ... ed applicativi (HTTP, HTTPS, FTP, ...)



::. Generic Connection Framework (GCF)

Connettività

- J2ME deve supportare una vasta gamma di dispositivi con differenti capacità di networking e requisiti di I/O
- E' richiesta un'architettura di I/O notevolmente flessibile
- Per tale scopo è stato definito un framework di classi "Generiche" atte a gestire connessioni indipendentemente dalla particolare tecnologia utilizzata
- Tale framework è stato denominato **GENERIC CONNECTION FRAMEWORK** ed inglobato nella configurazione CLDC dalla sua versione 1.0
- In seguito poi il GCF è stato incorporato anche nella configurazione CDC, e più recentemente nella piattaforma J2SE (JSR-197)



::. Generic Connection Framework (GCF)

Cosa è il GCF?

- E' una gerarchia di classi ed interfacce per creare connessioni eterogenee ed effettuare operazioni di I/O
(*definite nel package `javax.microedition.io`*)
- Fornisce un API comune a tutti i tipi di connessione
- Alcuni esempi di protocolli supportati:
 - btl2cap (bluetooth)
 - datagram (datagram)
 - file (accesso ad archivi locali)
 - http (hyperText transport protocol)
 - https
 - comm (comunicazioni seriali)
 - sms (Short Messaging Service)
 - mms
 - cbs (Cell Broadcast Service)



::. Generic Connection Framework (GCF)

I “building blocks”

- **Gerarchia di Interfacce Estensibile**

Fornisce Interfacce sia per connessioni di tipo Datagram che di tipo Stream

- **Factory di Connessioni**

Consente di creare una connessione utilizzando uno specifico schema

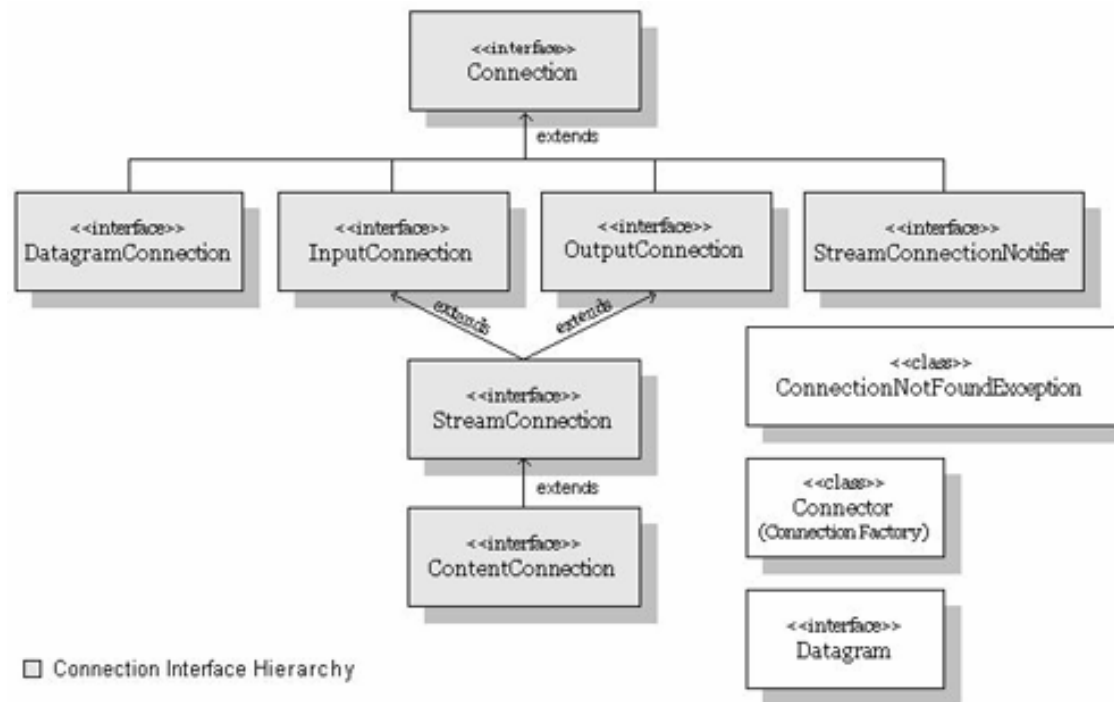
- **Standard Uniform Resource Locators (URLs)**

Fornisce un metodo per indicare quali connessioni creare



::: Generic Connection Framework (GCF)

Gerarchia di classi ed Interfacce



::: Generic Connection Framework (GCF)

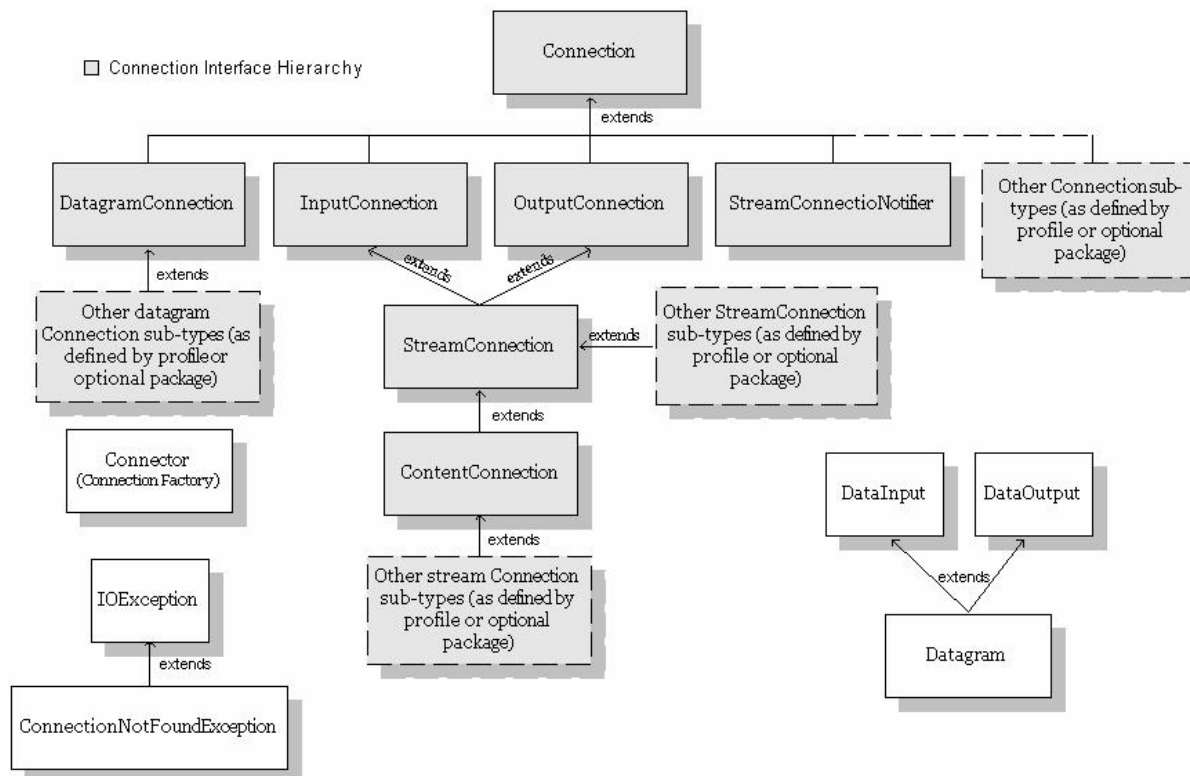
Classi ed Interfacce Principali

- **Connection** - Interfaccia base per tutti i tipi di connessione
- **DatagramConnection** - Invio e Ricezione Dati basata su pacchetti (non solo UDP, ma anche TCP)
- **InputConnection, OutputConnection** - Invio e Ricezione Dati basata su Streams
- **StreamConnection** - Connessione con flusso dati bidirezionale
- **ContentConnection** - Specializzazione della **StreamConnection**, aggiunge informazioni relative al contenuto dei dati ed alla loro codifica
- **StreamConnectionNotifier** - Utilizza per le notifiche in caso di comunicazioni asincrone
- **Connector** - Factory delle connessioni. Il suo metodo `open` richiede la URL della connessione da aprire (la URL deve includere schema e/o protocollo)



::: Generic Connection Framework (GCF)

Estensibilità



• GCF definisce il minimo comune denominatore tra i diversi tipi di connessione possibili

• ... le connessioni HTTP sono decisamente diverse da quelle Bluetooth!

• Nuovi tipi di connessione possono essere definiti estendendo così il framework. Queste connessioni possono essere specifiche per un determinato profilo o ancora essere contenute in package opzionale



::. Generic Connection Framework (GCF)

URLs

Seguono lo schema generale definito nelle RFC 1738 e 2396

scheme : `//user:password@host:port/url-path;parameters`

scheme – specifica il metodo di accesso o protocollo da utilizzare. In GCF specifica il tipo di connessione da utilizzare

user – nome utente (opzionale, da usare in contesti che richiedono autenticazione)

Password – password utente (opzionale, da usare in contesti che richiedono autenticazione)

host – nome FQDN o indirizzo IP della macchina sulla quale si trova la risorsa puntata dalla URL

port – porto da utilizzare (opzionale, il significato dipende dallo schema)

url-path – “percorso” per raggiungere la risorsa. Il formato dipende dallo schema e può essere corredato da parametri opzionali



::. Generic Connection Framework (GCF)

Creazione, Gestione e Chiusura di una connessione (1/4)

```
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;

public class GCFConn extends MIDlet {
    private Display display;
    private String url = "http://wpage.unina.it/saorland/helloCMS.txt";
    public GCFConn() {
        display = Display.getDisplay(this);
    }
    public void startApp() {
        try {
            downloadPage(url);
        } catch (IOException e) {
            alert = new Alert(message, e.toString(), null, AlertType.WARNING);
            alert.setTimeout(Alert.FOREVER);
            display.setCurrent(alert);
        }
    }
}
```

[CONTINUA ...]

www.mobilab.unina.it

info@mobilab.unina.it



Mobile Systems

:: Generic Connection Framework (GCF)

Creazione, Gestione e Chiusura di una connessione (2/4)

```
private void downloadPage(String url) throws IOException {  
    StringBuffer b = new StringBuffer();  
    StringBuffer h = new StringBuffer();  
    String hf = ""; // not null  
    InputStream is = null;  
    HttpURLConnection c = null;  
    TextBox t = null;  
    try {  
        long len = 0 ;  
        int ch = 0;  
        c = (HttpURLConnection)Connector.open(url);  
        is = c.openInputStream();  
        // Fetch the HTTP Header  
        for (int i = 0 ; hf != null; i++) {  
            hf = c.getHeaderFieldKey(i);  
            h.append(i+": "+hf);  
            hf = c.getHeaderField(i);  
            if (hf != null && hf != "")  
                h.append("="+hf+"\n");  
        }  
    }  
}
```

[CONTINUA ...]

www.mobilab.unina.it

info@mobilab.unina.it



Mobile Systems

::. Generic Connection Framework (GCF)

Creazione, Gestione e Chiusura di una connessione (3/4)

[CONTINUA METODO `downloadPage`]

```
len = c.getLength();
if ( len != -1) {
    // Read exactly Content-Length bytes
    for (int i =0 ; i < len ; i++ )
        if ((ch = is.read()) != -1)
            b.append((char) ch);
        else {
            // Read till the connection is closed.
            while ((ch = is.read()) != -1) {
                len = is.available() ;
                b.append((char) ch);
            }
        }
}
```

[CONTINUA ...]



::: Generic Connection Framework (GCF)

Creazione, Gestione e Chiusura di una connessione (4/4)

[CONTINUA METODO `downloadPage`]

```
// Dump file data to the device screen
t = new TextBox("hello again....",h.toString()+b.toString(),2048,0);
// Dump file data to kToolBar
System.out.println("Headers");
System.out.println(h.toString());
System.out.println("Body");
System.out.println(b.toString());
} finally {
    is.close();
    c.close();
}
display.setCurrent(t);
}

public void pauseApp() {}
public void destroyApp(boolean unconditional) {}

}
```



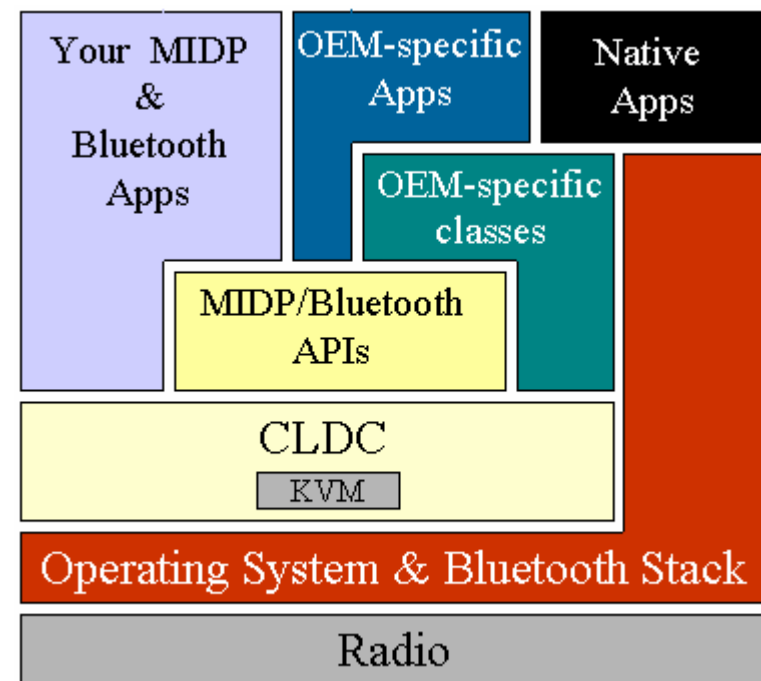
::: Java APIs for Bluetooth (JSR-82)

Prima API standard, aperta e non proprietaria per sviluppare applicazioni Bluetooth in Java.

La complessità dello stack protocollare bluetooth viene nascosta dietro un insieme di classi Java che nascondono i dettagli di basso livello di Bluetooth

JSR-82 è composta da 2 packages opzionali:

- 1) **javax.bluetooth**
Nucleo delle API Bluetooth
- 2) **javax.obex**
APIs per il protocollo "Object Exchange"
[indipendenti da Bluetooth]



::: Java APIs for Bluetooth (JSR-82)

Requisiti di sistema

- Memoria minima 512K (ROM e RAM)
- Connessione di rete Bluetooth
- Implementazione J2ME conforme a specifiche CLDC
- Requisiti per il sottosistema Bluetooth:
 - Generic Access Profile
 - Service Discovery Application Profile
 - Serial Port Profile
 - Service Discovery Protocol (**SDP**)
 - Radio Frequency COMMunication protocol (**RFCOMM**)
 - Logical Link Control and Adaptation Protocol (**L2CAP**)
 - Bluetooth Control Center (**BCC**)



::: Java APIs for Bluetooth (JSR-82)

Capabilities delle API JSR-82

- Registrazione di servizi Bluetooth
- Ricerca di dispositivi e servizi (offerti da altri dispositivi)
- Stabilire connessioni RFCOMM, L2CAP ed OBEX con altri dispositivi
- Effettuare I/O su tali connessioni
- Gestire e controllare le connessioni
- Fornire la "security" per tutte queste operazioni
- Opera sia con stack bluetooth nativi che completamente scritti in Java. Nel primo caso le API fanno affidamento sull'interfaccia tra la JVM e lo stack nativo (ad esempio via KNI)



::: Java APIs for Bluetooth (JSR-82)

Sviluppare Applicazioni Bluetooth

Lo sviluppo di un applicazione Java Bluetooth può essere articolato in 6 fasi:

- 1. Inizializzazione dello stack protocollare**
- 2. Gestione dei dispositivi**
- 3. Ricerca Dispositivi**
- 4. Ricerca Servizi**
- 5. Registrazione del servizio**
- 6. Comunicazione**



::: Java APIs for Bluetooth (JSR-82)

Inizializzazione dello stack protocollare

- Lo stack Bluetooth ha la responsabilità di controllare l'interfaccia di rete
- Lo stack Bluetooth deve essere inizializzato prima di effettuare qualunque operazione
- Scopo dell'inizializzazione è rendere il dispositivo pronto a comunicare
- A tal proposito deve essere utilizzato il control center (BCC)
 - Implementazione vendor-specific
(stack differenti si inizializzano in maniera differente)
 - Le API per la gestione del BCC non fanno parte delle JSR-82
(possono variare da vendor a vendor)
 - Alcuni stack protocollari non dispongono di alcuna API per l'inizializzazione



::: Java APIs for Bluetooth (JSR-82)

Gestione dei dispositivi

- Effettuata mediante le classi LocalDevice and RemoteDevice
- LocalDevice rappresenta il dispositivo locale
consente di ottenere informazioni circa il tipo di dispositivo ed i servizi che offre

```
...  
// retrieve the local Bluetooth device object  
LocalDevice local = LocalDevice.getLocalDevice();  
// retrieve the Bluetooth address and name of the local device  
String address = local.getBluetoothAddress();  
String name = local.getFriendlyName();  
...
```

- RemoteDevice rappresenta un dispositivo remoto
consente di ottenere informazioni sul dispositivo come indirizzo e nome

```
...  
// retrieve the device that is at the other end of the Bluetooth connection  
RemoteDevice remote = RemoteDevice.getRemoteDevice(Connection c);  
// retrieve the Bluetooth address and name of the remote device  
String remoteAddress = remote.getBluetoothAddress();  
String remoteName = local.getFriendlyName(true);  
...
```



::: Java APIs for Bluetooth (JSR-82)

Ricerca dei Dispositivi (1/2)

- **I Dispositivi sono mobili: prima di effettuare la ricerca è necessario effettuare uno SCAN per vedere quali siano disponibili nel range del proprio dispositivo**
- **Realizzato tramite la classe `DiscoveryAgent` e l'interfaccia `DiscoveryListener`**
- **Mediante `DiscoveryAgent` vengono cercati i dispositivi raggiungibili (metodo `startInquiry`)**
- **Ogni qual volta un dispositivo viene individuato viene invocato il metodo `deviceDiscovered` dell'interfaccia `DiscoveryListener`**
- **Al termine delle operazioni di ricerca viene invocato il metodo `inquiryCompleted` di `DiscoveryListener`**
- **`DiscoveryAgent` possiede una "cache" di dispositivi "noti", reperibili mediante il metodo `retrieveDevices`**
 - *Elenco di dispositivi "noti a priori" (frequentemente contattati)*
 - *Elenco di dispositivi trovati in inquiries precedenti*



::: Java APIs for Bluetooth (JSR-82)

Ricerca dei Dispositivi (2/2)

- **DEVICE INQUIRY**

```
...  
// retrieve the discovery agent  
DiscoveryAgent agent = local.getDiscoveryAgent();  
// place the device in inquiry mode  
boolean complete = agent.startInquiry();  
...
```

- **RECUPERO LISTA DISPOSITIVI NOTI**

```
...  
// retrieve the discovery agent  
DiscoveryAgent agent = local.getDiscoveryAgent();  
// return an array of pre-known devices  
RemoteDevice[] devices =  
agent.retrieveDevices(DiscoveryAgent.PREKNOWN); //usare CACHED per  
//risultati inquiries precedenti  
...
```



::: Java APIs for Bluetooth (JSR-82)

Ricerca dei Servizi

- Dopo aver individuato un dispositivo, è buona norma sapere quale servizi offre
- Il dispositivo locale avvia la ricerca dei servizi sul dispositivo remoto, sempre mediante `DiscoveryAgent`
- **Attenzione:** la API JSR-82 offre metodi per cercare servizi sul dispositivo remoto, ma non per cercare servizi sul dispositivo locale



::: Java APIs for Bluetooth (JSR-82)

Registrazione Servizi

- Affinchè un servizio possa venir localizzato da un dispositivo deve essere prima registrato
- La registrazione deve essere effettuata su un Bluetooth server device
 1. Creazione del record che descrive il servizio offerto
 2. Aggiunta del record di servizio al "Server Service Discovery Database" (SDBB)
 3. Registrazione delle misure di sicurezza associate al servizio
- Il server deve inoltre:
 - Accettare connessioni dai clients
 - Aggiornare il record di servizio nel SDBB se richiesto
 - Rimuovere o disabilitare il record di servizio nel SDBB se richiesto



::: Java APIs for Bluetooth (JSR-82)

Comunicazione via Bluetooth

- Le JSR-82 forniscono supporto per i seguenti protocolli:
 - **L2CAP** – La parte data-link dello stack Bluetooth che fornisce sia l'astrazione di comunicazione connection-oriented che quella connectionless. E' inoltre responsabile per le seguenti operazioni:
 1. Multiplexing e demultiplexing del canale
 2. Segmentation And Reassembly
 - **RFCOMM** – emulazione porta seriale RS232 su Bluetooth, implementato sul protocollo L2CAP.
 - **OBEX** – Protocollo per lo scambio di oggetti (tipicamente files) su reti Wireless (non solo Bluetooth!)



::: Java APIs for Bluetooth (JSR-82)

Serial Port Profile (1/2)

- RFCOMM costruito su L2CAP
- Emula comportamento porta seriale (RS-232)
- Solo 1 connessione RFCOMM alla volta tra due dispositivi
- Max 60 connessioni seriali "logiche" possono essere mappate sulla connessione RFCOMM
- Un dispositivo Bluetooth può avere max 30 servizi RFCOMM (COM1... COM30)
- Un dispositivo può supportare solo una determinata connessione RFCOMM con un determinato client alla volta



::: Java APIs for Bluetooth (JSR-82)

Serial Port Profile (2/2)

- **Ruolo del server nella connessione:**
 - **Costruire una URL che indica come connettersi al servizio. Memorizzare tale URL nel SDDB**
 - **Rendere il *service record* disponibile ai clients**
 - **Accettare una connessione da un client**
 - **Scambiare dati con il client (ovviamente !)**
- **Esempio di URL:**

btsp://102030405060740A1B1C1D1E100:5

Proto

Address

Port (porta COM)



:: Java APIs for Bluetooth (JSR-82)

Esempio Serial Port Profile (1/2) - *Server*

```
...
// service URL
String ServiceURL =
    "btspp://localhost:10203040607040A1B1C1DE100;name=SPP
    Server1";
try {
    // create a server connection
    StreamConnectionNotifier notifier =
        (StreamConnectionNotifier) Connector.open(serviceURL);
    // accept client connections
    StreamConnection connection = notifier.acceptAndOpen();
    // prepare to send/receive data
    byte buffer[] = new byte[100];
    String msg = "hello there, client";
    InputStream is = connection.openInputStream();
    OutputStream os = connection.openOutputStream();
    // send data to the client
    os.write(msg.getBytes());
    // read data from client
    is.read(buffer);
    connection.close();
} catch (IOException e) {
    e.printStackTrace();
}
...
```



:: Java APIs for Bluetooth (JSR-82)

Esempio Serial Port Profile (2/2) - *Client*

```
...  
// (assuming we have the service record)  
// use record to retrieve a connection URL  
String url =  
    record.getConnectionURL(  
        record.NOAUTHENTICATE_NOENCRYPT, false);  
// open a connection to the server  
StreamConnection connection =  
    (StreamConnection) Connector.open(url);  
// Send/receive data  
try {  
    byte buffer[] = new byte[100];  
    String msg = "hello there, server";  
    InputStream is = connection.openInputStream();  
    OutputStream os = connection.openOutputStream();  
    // read data from the server  
    is.read(buffer);  
    // send data to the server  
    os.write(msg.getBytes());  
    connection.close();  
} catch (IOException e) {  
    e.printStackTrace();  
}  
...
```

