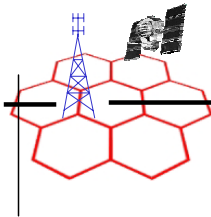




Università degli Studi di Roma “La Sapienza”

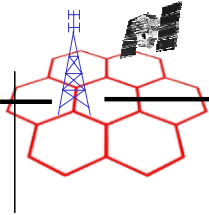
Dipartimento di Informatica e Sistemistica



Application Development on Constrained Devices

**Reti Wireless, a.a. 2004/2005**

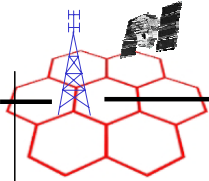
Emiliano Trevisani



## Contatti

---

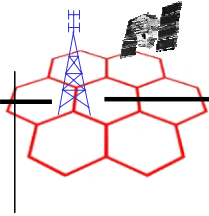
- Emiliano Trevisani:
  - Via Salaria 113, Stanza C2
  - [emiliano.trevisani@dis.uniroma1.it](mailto:emiliano.trevisani@dis.uniroma1.it)
- Lucidi e links utili su <http://www.dis.uniroma1.it/~trevisani>



# Summary

---

- ❑ **Overview**
- ❑ **Java 2 ME**
  - **Introduzione**
  - *Java2 xx VS Java2 ME*
  - **Configurazioni e Profili**
  - **MIDLET & API set**
  - **Esempi di MIDLET**
  - **Evoluzioni**
- ❑ **Symbian OS**
  - **Introduzione**
  - **API set**
  - **Esempio di applicazione a console**

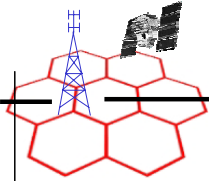


## Java 2 ME - Introduzione

---

### □ Storia:

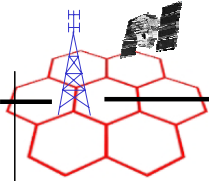
- Inizio anni '90: **OAK** [Green Project]: linguaggio OO per dispositivi di consumo (“*write once, run anywhere*” .....
- 1996: **Java Card**: piattaforma Java per Smart Card
- 1997: **PersonalJava**: dispositivi connessi alla rete, con interfaccia utente
- 1998: **EmbeddedJava**: dispositivi Java integrati: ogni costruttore sceglie le API adatte per la sua applicazione
- 1999: **Spotless System e KVM**: la più piccola JVM con le funzionalità Java di base



## Java 2 ME - Introduzione

---

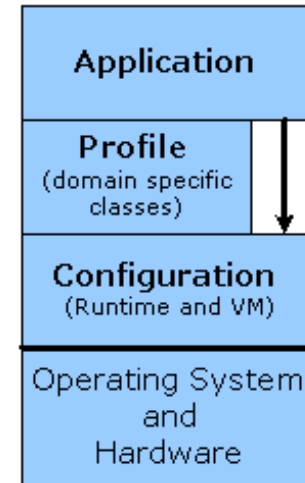
- Task:
  - Struttura della piattaforma
  - Le configurazioni
  - Le Virtual Machines
  - I profili
  - Packages opzionali

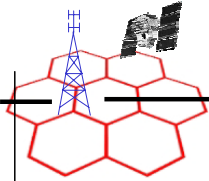


## Java 2 ME - Introduzione

---

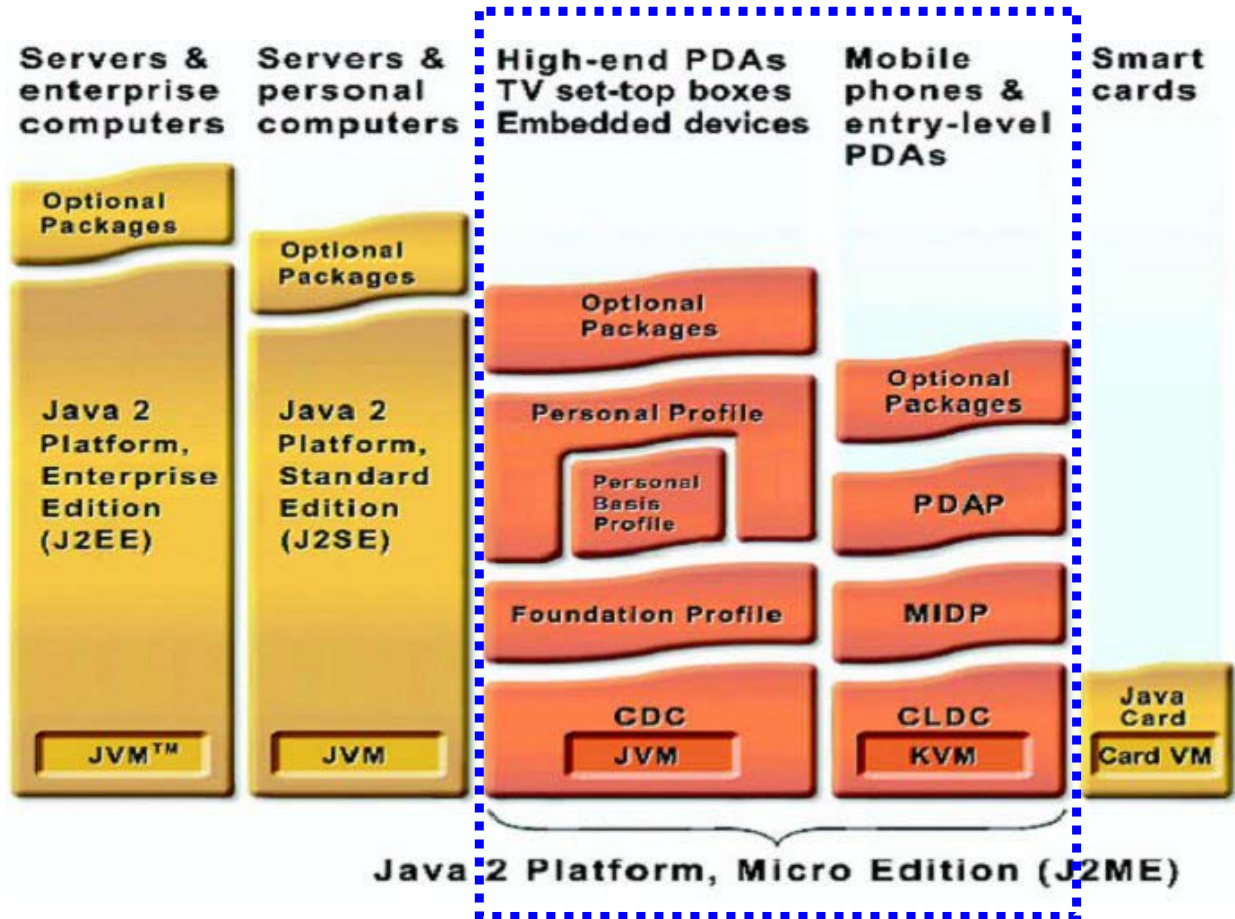
- ❑ Idea base: “adattare” Java tradizionale alle caratteristiche dei dispositivi mobili
- ❑ Struttura J2ME: Configurazioni
  - **CLDC**, con relativi profili
  - **CDC**, con relativi profili

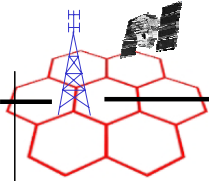




# Java 2 ME - Java2 xx VS Java2 ME

Struttura  
della  
piattaforma

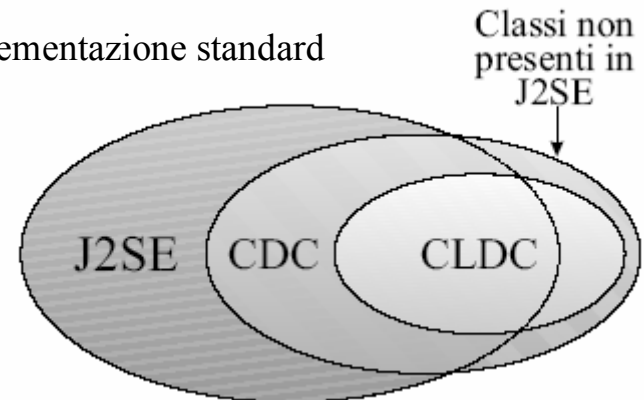


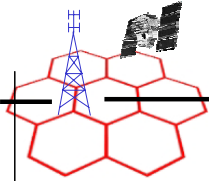


## Java 2 ME - Java2 xx VS Java2 ME

---

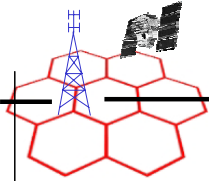
- ❑ Una **configurazione** definisce l'ambiente runtime J2ME di base: include la VM ed un set di API fondamentali derivate da J2SE
- ❑ Virtual Machines
  - **KVM (K VM) per CLDC**
    - VM compatta creata appositamente per dispositivi dalle risorse limitate.
    - Occupa 32 - 80 KB
  - **CVM (Compact VM) per CDC**
    - VM con pieno supporto J2SE 1.3
    - Più portatile, efficiente, ridotta rispetto all'implementazione standard





## Java 2 ME – Configurazioni e Profili

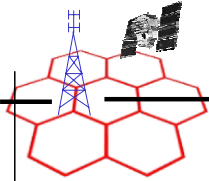
Configurazione	Descrizione	Dispositivi Target
<b>CLDC</b> [Connected Limited Device Configuration]	Adatta a dispositivi di consumo con limitate capacità. Il nucleo è la Virtual Machine <i>KVM</i>	Processore a 16-32 bit, almeno 160KB di memoria persistente, almeno 32K non pers., qualche tipo di connessione (es. cellulari, low-end PDAs)
<b>CDC</b> [Connected Device Configuration]	Per dispositivi con capacità meno restrittive dei precedenti. Il nucleo è la Virtual Machine <i>CVM</i>	Processore a 32 bit, almeno 2MB di memoria totale, qualche tipo di connessione (es. high-end PDAs, disp. embedded avanzati)



## Java 2 ME - Configurazioni e Profili

---

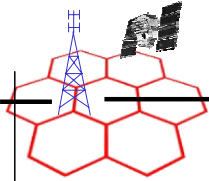
- ❑ Un **profilo** estende la configurazione base, aggiungendo classi adatte al dominio applicativo
- ❑ Profili CLDC:
  - MIDP: adatto allo sviluppo di applicazioni per dispositivi wireless come cellulari e smartphone
  - PDAP: profilo creato appositamente per PDA, estende MIDP e CLDC per sfruttare le maggiori potenzialità di questi dispositivi
- ❑ Profili CDC:
  - FP – Foundation Profile: profilo di più basso livello, adatto a periferiche integrate senza interfaccia utente (non fornisce infatti supporto GUI). È la base per gli altri profili CDC.
  - PBP – Personal Basis Profile: estende FP con un sottoinsieme di API AWT per dispositivi con semplici interfacce grafiche.
  - PP – Personal Profile: estende FP con tutte le API AWT e supporto per Applet. Rappresenta la nuova implementazione di Personal Java.



## Java 2 ME - Midlet & API Set

---

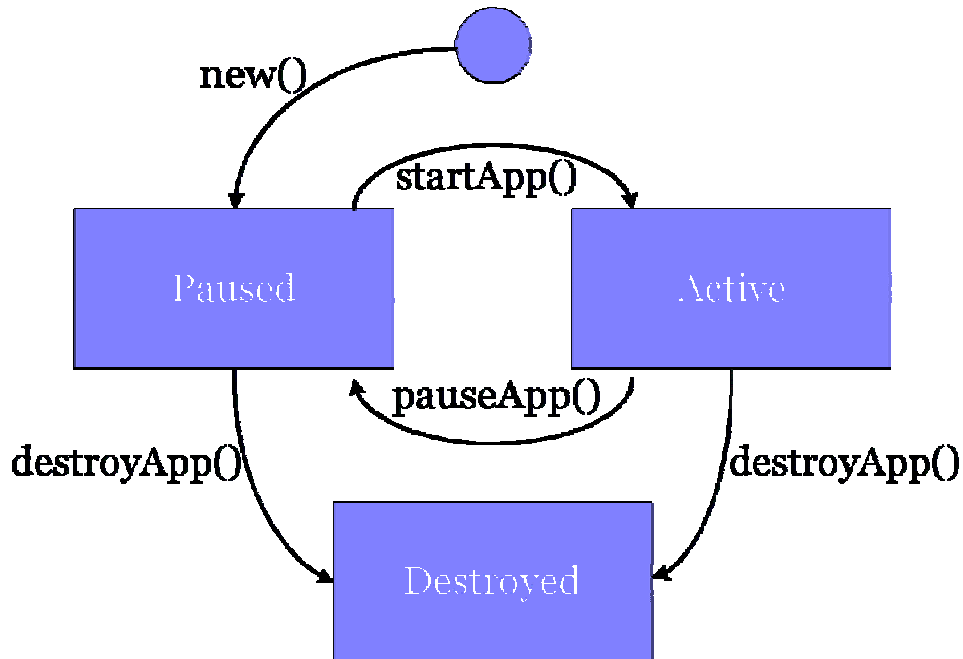
- Packages Opzionali [oltre alla configurazione + profilo]:
  - Java API per Bluetooth
  - Wireless Messaging API (WMA 1.1, 1.2,..)
  - Mobile Media API (MMAPI)
  - Package RMI
  - JDBC per FP
  - Location API
  - ...anche di terze parti

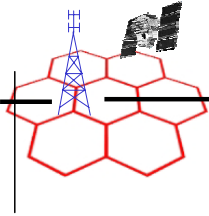


## Java 2 ME – Midlet & API Set

---

- ❑ Una MIDLET è un'applicazione definita nel profilo MIDP;
- ❑ Il ciclo di vita di una Midlet (caricamento – esecuzione – distruzione) si ispira al modello utilizzato nelle java Applet:



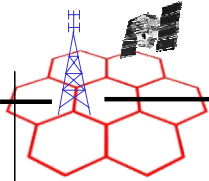


## Java 2 ME – Midlet & API Set

---

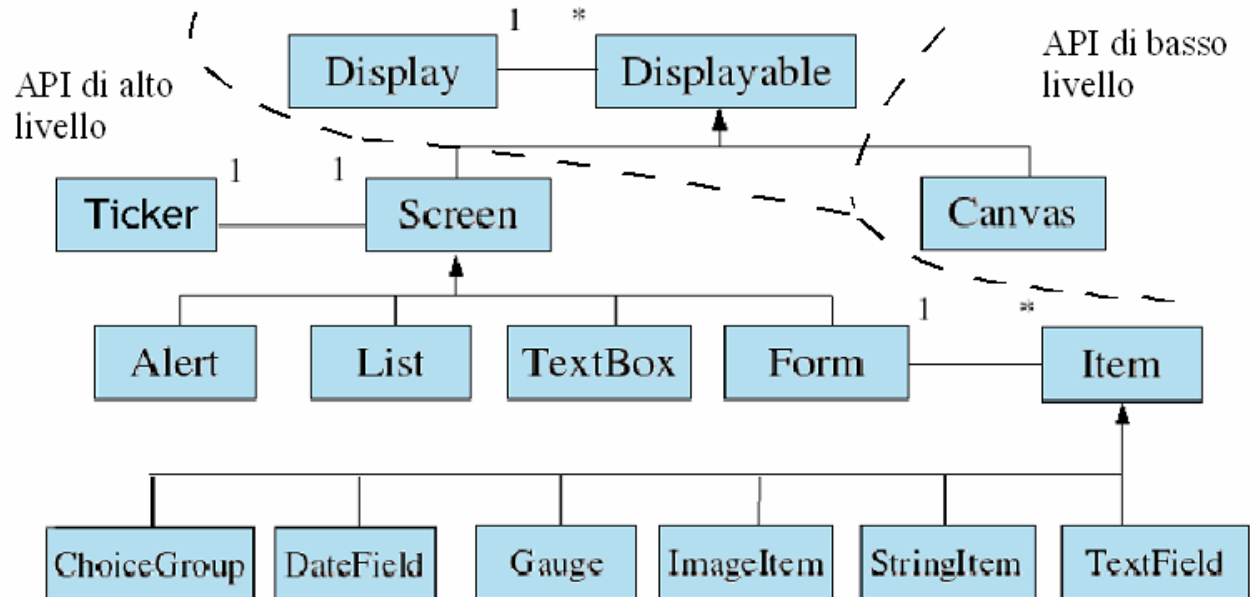
### □ Package MIDP 1.0

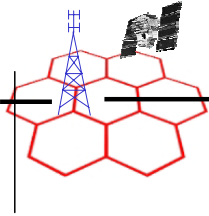
- **java.lang**: classi base, come in J2SE (*Integer, Byte, Thread, ...*). Mancano *Double, Float, ThreadGroup, Process, ...*
- **java.util**: classi d'utilità, come J2SE (*Calendar, Date, ...*). Mancano *Collection, Enumeration, List, Map, ...*
- **java.io**: I/O da un generico stream di dati, come J2SE
- **javax.microedition.midlet**: package base per le MIDlet
- **javax.microedition.lcdui**: GUI
- **javax.microedition.rms**: storage persistente dei dati
- **javax.microedition.io**: networking



## Java 2 ME – Midlet & API Set

### □ javax.microedition.lcdi





## Java 2 ME – Midlet & API Set

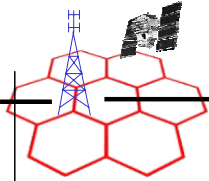
---

### □ Display

- rappresenta il display fisico del dispositivo
- **unico**: vi si accede tramite il metodo statico *getDisplay* della classe omonima
- alcuni metodi:
  - *setCurrent(Displayable), getCurrent()*
  - *isColor(), getNumColors(), ...*

### □ Displayable

- contenitore di ciò che viene visualizzato (schermata)
- l'applicazione può utilizzare *n Displayable*, ma ne visualizza uno solo alla volta;
- alcuni metodi:
  - *isShown(), ...*



## Java 2 ME – Primo esempio di Midlet 1/2

---

```
package simpleMidlet;
```

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

```
import javax.microedition.lcdui.*;
```

```
public class SimpleMidlet_1 extends MIDlet
```

```
{
```

```
    private Display display;
```

```
    private Displayable view;
```

```
    public void destroyApp(boolean unconditional)
```

```
    {
        notifyDestroyed();
    }
```

```
    public void pauseApp()
```

```
    {
```

```
    }
```

```
    public void startApp()
```

```
    {
```

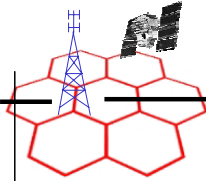
```
        display = Display.getDisplay(this);
```

```
        view = new Form("I'm HELLO Form");
```

```
        display.setCurrent(view);
```

```
    }
```

```
}
```



## Java 2 ME – Primo esempio di Midlet 2/2

```
package simpleMidlet;
```

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

*Ogni applicazione importa almeno il package javax.microedition.midlet; gli altri package a seconda delle necessità.*

```
public class SimpleMidlet_1 extends MIDlet  
{
```

```
    private Display display;  
    private Displayable view;
```

*La classe base dell'applicazione deve ereditare javax.microedition.Midlet*

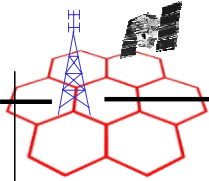
```
    → public void destroyApp(boolean unconditional)  
        {  
            notifyDestroyed();  
        }
```

```
    → public void pauseApp()  
        {  
        }
```

```
    → public void startApp()  
        {
```

```
            display = Display.getDisplay(this);  
            view = new Form("I'm HELLO Form");  
            display.setCurrent(view);
```

```
    }
```



## Java 2 ME – Secondo esempio di Midlet 1/3

---

```
package simpleMidlet;
```

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

```
import javax.microedition.lcdui.*;
```

```
public class SimpleMidlet_2 extends MIDlet implements CommandListener
```

```
{
```

```
    private Display display;
```

```
    private Form view;
```

```
    public void destroyApp(boolean unconditional)
```

```
    {
        notifyDestroyed();
    }
```

```
    public void pauseApp()
```

```
    {
```

```
    }
```

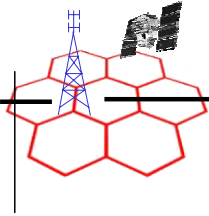
```
    public void startApp()
```

```
    {
```

```
        display = Display.getDisplay(this);
```

```
        view = new Form("I'm HELLO Form");
```

```
        .....
```



## Java 2 ME – Secondo esempio di Midlet 2/3

---

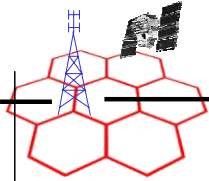
```
TextField userIdField=new TextField("User ID", "",8,TextField.NUMERIC);  
TextField userPwdField=new TextField("Password", "",8,TextField.PASSWORD);
```

```
Command okCommand = new Command( "Login",Command.OK, 1 );  
Command exitCommand = new Command( "Exit",Command.CANCEL, 2);
```

```
view.append(userIdField);  
view.append(userPwdField);  
view.addCommand(okCommand);  
view.addCommand(exitCommand);  
view.setCommandListener(this);
```

```
display.setCurrent(view);
```

```
}
```



## Java 2 ME – Secondo esempio di Midlet 3/3

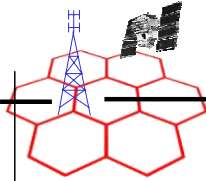
---

```
public void commandAction(Command c, Displayable d)
{
    switch (c.getPriority())
    {
        case 1:    Form currentForm=(Form)d;
                  Alert infoMsg=new Alert("Your input: UserID="+((TextField)currentForm.get(0)).getString()+",
                  PWD="+((TextField)currentForm.get(1)).getString());
                  infoMsg.setType(AlertType.INFO);
                  infoMsg.setTimeout(Alert.FOREVER);

                  display.setCurrent(infoMsg);
                  break;

        case 2:    System.exit(0);
                  break;
    }
}
}
```

*L'oggetto Displayable corrente [view] viene sostituito con l'oggetto Displayable infoMsg*



## Java 2 ME – Terzo esempio di Midlet 1/10

---

```
package j2medeveloper.wma;
```

```
import java.util.*;
```

```
import java.io.IOException;
```

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

```
import javax.microedition.lcdui.*;
```

```
import javax.microedition.io.*;
```

```
import javax.wireless.messaging.*;
```

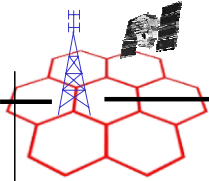
**MIDP 1.0 + WMA 1.0**

```
public class WMAMIDlet extends MIDlet implements MessageListener, CommandListener  
{
```

```
// Message connections
```

```
MessageConnection mcs, mcc; // One server and one client connection
```

```
static boolean _singlepass = true;
```



## Java 2 ME – Terzo esempio di Midlet 2/10

---

// Display environment information

```
private Command serverCmd = new Command( "Start SMS Server",Command.SCREEN, 1 );
```

```
private Command sendCmd = new Command( "Send SMS",Command.SCREEN, 2 );
```

```
private Command okNumberCmd = new Command( "OK",Command.OK, 1 );
```

```
private Command cancelNumberCmd = new Command( "Cancel",Command.CANCEL, 2 );
```

```
private TextField inputField;
```

```
protected Display display;
```

```
private Ticker ticker;
```

```
private List list;
```

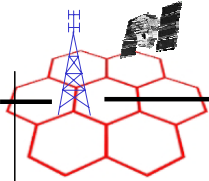
```
private Form mainForm;
```

/\*\* Constructor \*/

```
public WMAMIDlet() {  
}
```

/\*\* Initial state \*/

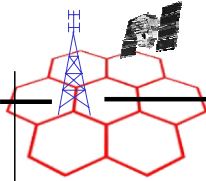
```
public void startApp() throws MIDletStateChangeException  
{
```



## Java 2 ME – Terzo esempio di Midlet 3/10

---

```
try {    if (display == null) { // UI related.
        String smsc = System.getProperty("wireless.messaging.sms.smsc");
        display = Display.getDisplay(this);
        list = new List("WMA MIDLET Test", List.IMPLICIT);
        list.addCommand(serverCmd);
        list.addCommand(sendCmd);
        list.setCommandListener(this);
        ticker = new Ticker("");
        list.setTicker(ticker);
        inputField=new TextField("PhoneNumber", "", 15, TextField.PHONENUMBER);
        mainForm = new Form("Insert Phone Number");
        mainForm.append(inputField);
        mainForm.addCommand(okNumberCmd);
        mainForm.addCommand(cancelNumberCmd);
        mainForm.setCommandListener(this);
        display.setCurrent(list); } //if
    }
    catch (Exception e) {
        // If some kind of error occurred, throw MIDlet exception.
        throw new MIDletStateChangeException("Error Starting");
    }
} // startApp()
```



## Java 2 ME – Terzo esempio di Midlet 4/10

---

```
/** Paused state.*/
```

```
public void pauseApp() {}
```

```
/**
```

Destroy state. Release resources (connection, threads, etc).

@param uc If true when this method is called, the MIDlet must cleanup and release all resources. If false the MIDlet may throw MIDletStateChangeException to indicate it does not want to be destroyed at this time.

```
*/
```

```
public void destroyApp(boolean uc) throws MIDletStateChangeException {
```

```
    try {
```

```
        if (mcs != null) {
```

```
            mcs.setMessageListener(null);
```

```
            mcs.close();
```

```
        }
```

```
        if (mcc != null) {
```

```
            mcc.close();
```

```
        }
```

```
        display = null;
```

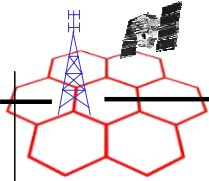
```
    }
```

```
    catch (IOException e) {
```

```
        // Handle the exception...
```

```
    }
```

```
}
```

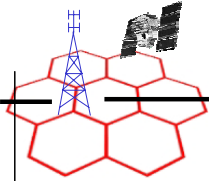


## Java 2 ME – Terzo esempio di Midlet 5/10

---

```
/**
 * newMessageConnection returns a new MessageConnection
 * @param addr is the address (local or remote)
 * @return MessageConnection that was created
 * @throws Exception if an error is encountered
 */
public MessageConnection newMessageConnection(String addr) throws Exception {
    return((MessageConnection)Connector.open(addr));
}

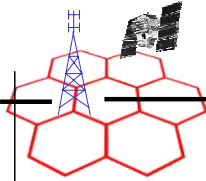
/**
 * Asynchronous callback for inbound message.
 * @param conn the MessageConnection with incoming Message
 */
public void notifyIncomingMessage(MessageConnection conn) {
    // Dispatch a message processor to process the incoming message
    MessageProcessor mp = new MessageProcessor(conn, _singlepass);
}
```



## Java 2 ME – Terzo esempio di Midlet 6/10

---

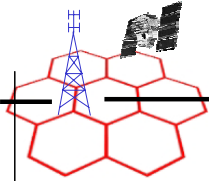
```
/**
 * sendTextMessage sends a TextMessage on the specified connection
 * @param mc the MessageConnection
 * @param msg the message to send
 * @param url the destination address, typically used in server mode
 */
public void sendTextMessage(MessageConnection mc, String msg, String url) {
    try {
        TextMessage tmsg =
            (TextMessage)mc.newMessage(MessageConnection.TEXT_MESSAGE);
        if (url != null)
            tmsg.setAddress(url);
        tmsg.setPayloadText(msg);
        int segcount = mc.numberOfSegments(tmsg);
        if (segcount == 0) {
            // Alert User!
        }
        else
            mc.send(tmsg);
    }
    catch (Exception e) { // Handle the exception...}
}
```



## Java 2 ME – Terzo esempio di Midlet 7/10

---

```
public void commandAction(Command c, Displayable d) {
    if (c == serverCmd ) { // Create a server SMS MessageConnection
        if (mcs == null) {
            try { mcs = new MessageConnection("sms://:5000"); // Open the messaging inbound port.
                MessageProcessor mp = new MessageProcessor(mcs); // Create a message processor
                mcs.setMessageListener(this); // Register a listener for inbound messages.
            }
            catch (Exception e) { // Handle Exception... }
        } //if } //if
    } else {
        if (c == sendCmd ) { display.setCurrent(mainForm); //Insert destination number }
        else if (c == okNumberCmd)
            { // Create a client SMS MessageConnection and send a message
                if (mcc == null) {
                    try { // Open the messaging inbound port.
                        mcc = new MessageConnection("sms://+39"+inputField.getString()+":5000");
                    }
                    catch (Exception e) { // Handle Exception... }
                } //if
                sendTextMessage(mcc, "Hello DIS !", null);
            }
        else display.setCurrent(list); } // commandAction()
    }
```



## Java 2 ME – Terzo esempio di Midlet 8/10

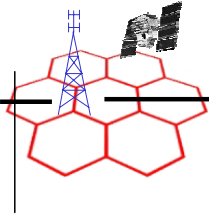
---

// MessageProcessor Inner class – This thread is responsible of receiving and processing MessageConnection messages.

```
class MessageProcessor implements Runnable {
    Thread th = new Thread(this);
    MessageConnection mc; // MessageConnection to handle
    boolean done; // if true, thread must exit
    boolean singlepass; // if true, read and process 1 message only and exit

    /** Constructor for multi or single-pass MessageProcessor
     * @param mc is the MessageConnection for this message processor
     * @param singlepass if true this processor must process only one message and exit
     */
    public MessageProcessor(MessageConnection mc, boolean singlepass) {
        this.mc = mc;
        this.singlepass = singlepass;
        th.start();
    }

    /** Notify the message processor to exit */
    public void notifyDone() {
        done = true;
    }
}
```



## Java 2 ME – Terzo esempio di Midlet 9/10

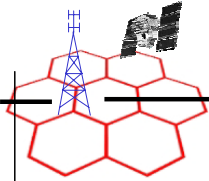
---

/\*\* Thread's run method to wait for and process received messages.\*/

```
public void run() {  
    if (singlepass == true)  
        processMessage();  
    else  
        loopForMessages();  
}
```

/\*\* Loop for messages until done \*/

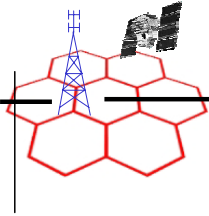
```
public void loopForMessages() {  
    while (!done)  
        processMessage();  
}
```



## Java 2 ME – Terzo esempio di Midlet 10/10

---

```
/** processMessage handles a received Message */
public void processMessage() {
    Message msg = null;
    // Try reading (maybe block for) a message
    try { msg = mc.receive(); }
    catch (Exception e) { // Handle reading errors }
    // Process the received message
    if (msg instanceof TextMessage) {
        TextMessage tmsg = (TextMessage)msg;
        ticker.setString(tmsg.getPayloadText()); // Handle the text message... }
    else {
        // process received message
        if (msg instanceof BinaryMessage) {
            BinaryMessage bmsg = (BinaryMessage)msg;
            byte[] data = bmsg.getPayloadData();
            // Handle the binary message...
        } else { // Ignore }
    }
}
} // MessageProcessor
} // WMAMIDlet
```

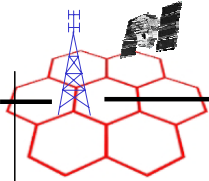


## Java 2 ME – Evoluzioni

---

- ❑ La possibilità di intercettare messaggi SMS entranti è una funzionalità già offerta in MIDP 1.0 ed esattamente in WMA 1.0 [Wireless Messaging API].
- ❑ Il servizio Push Registry è una novità introdotta in MIDP 2.0; tuttavia la possibilità di combinare l'esecuzione di MIDLET a seguito di SMS di trigger è offerta solo a partire dalle WMA 1.1 disponibili, in combinazione con MIDP 2.0, attualmente solo su pochissimi telefoni.
- ❑ Dal sito della SUN: <http://developers.sun.com/techttopics/mobility/midp/articles/pushreg/>

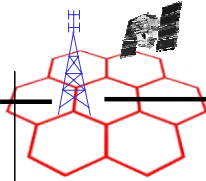
**".....MIDP 2.0 defines new connection types beyond HTTP, including TCP sockets and UDP datagrams, that can be set up as inbound connections, making them suitable for push. In addition, the Wireless Messaging API (WMA, JSR 120 [=WMA 1.1]) makes SMS-based push activation possible as well. The MIDP 2.0 specification doesn't mandate which inbound connection types must be available for push; that decision is left up to the platform vendors. Typically, though, we will have socket and datagrams available to us. Note that requesting an unsupported connection type will result in a ConnectionNotFoundException. ...."**



## Symbian OS - Introduzione

---

- ❑ Symbian OS is the advanced, open, standard operating system licensed by the world's leading mobile phone manufacturers. Symbian OS integrates the power of computing with telephony, bringing advanced data services – using voice, messaging and on-board processing power – to the mass market.
- ❑ Symbian OS enables mobile phones to be a platform for deployment of applications and services (programs and content) developed in a wide range of languages and media.
- ❑ Symbian OS is the common core of application programming interfaces (APIs) and technology that is shared by all Symbian OS phones. Symbian OS includes a multi-tasking kernel, middleware for communications, data management and graphics, the lower levels of the GUI framework, and application engines.
- ❑ By enabling flexible UI design on Symbian OS, Symbian is able to offer choice for manufacturers, carriers, enterprises and end-users. Using the same core operating system in different designs also eases application porting.
- ❑ As an operating system for integrated wireless communication, Symbian OS is the basis of the next generation of mobile phones.

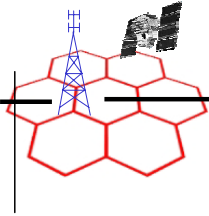


# Symbian OS - Introduzione

---

## Key features of Symbian OS:

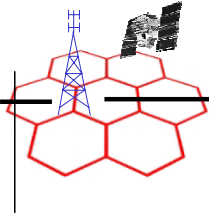
- ❑ comprehensive integration of contacts information, messaging, browsing and wireless telephony
- ❑ messaging – Internet email using POP3, IMAP4, SMTP, MHTML; standard attachments including Microsoft Word documents; fax; text messaging using SMS
- ❑ mobile telephony protocols – 2G voice and circuit-switched data, 2.5G packet-switched data, and SMS
- ❑ communication protocols – TCP/IP, WAP, Bluetooth®, IrDA, serial
- ❑ security – full-strength encryption and certificate management, secure communications protocols (including HTTPS, WTLS and SSL), certificate-based application installation
- ❑ rich suite of application engines, including contacts, schedule, messaging, browsing, voice, office, utility and system control
- ❑ object exchange – OBEX is used to exchange objects such as appointments and business cards
- ❑ multimedia server – support for several audio and image formats
- ❑ international locale support with Unicode characters, flexible text input framework, handwriting recognition, and additional font and text formatting support (supporting the Unicode Consortium standard)
- ❑ four main programming and content development options – C++, Java (including JavaPhone), WAP, and web
- ❑ data synchronization with PC-based applications using Symbian Connect
- ❑ support for multiple user interfaces – keyboard-based and/or pen-based information-centric mobile phones, and advanced data-enabled mobile phones



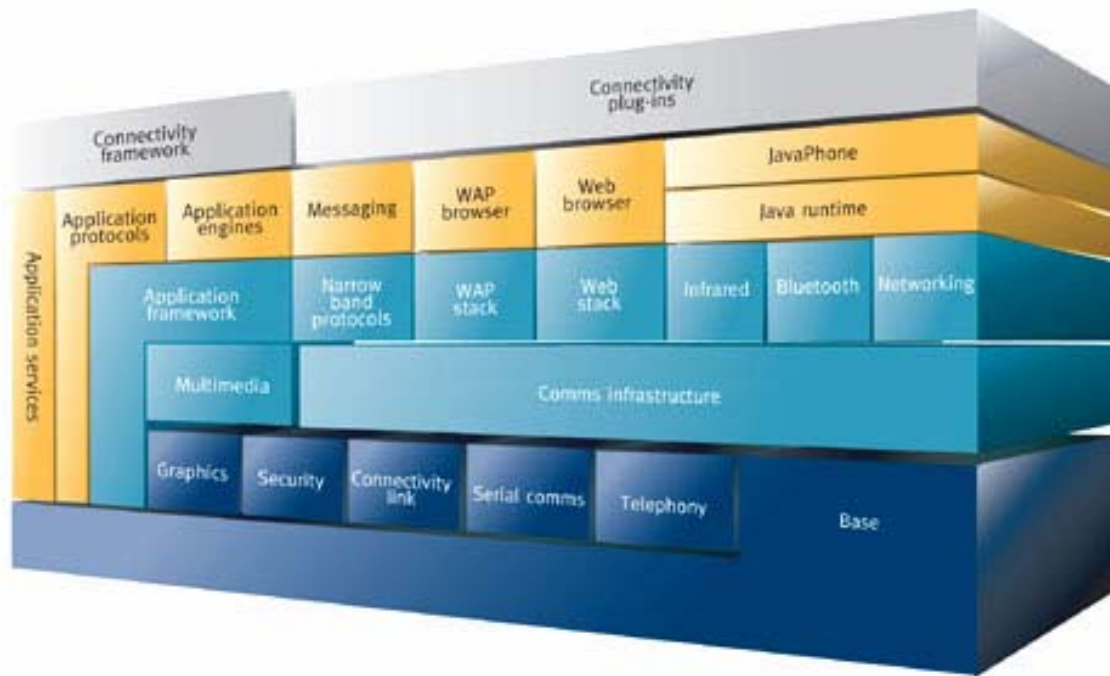
## Symbian OS - Architettura

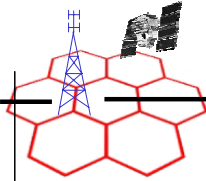
---

- Symbian OS is designed for the specific requirements of open, advanced, data-enabled mobile phones. Although compact enough to fit in the memory of a mobile phone, Symbian OS was planned from the beginning to be a full operating system in terms of functionality. The primary requirements are:
  - superior wireless communication integrated with personal data, applications on the phone, and data on other devices;
  - reliable handling of user data, even in the presence of unreliable communication, and shortage of resources such as memory, storage and power
  - to provide powerful, robust services
  - efficient use of all machine resources – especially power and memory
  - instant access to user data
  - support of industry standards
  - adaptability to mobile phone manufacturers' specific phone hardware and telephony stacks
  - the following diagram shows the basic architecture of Symbian OS Version 6.x and depicts system dependency rather than API usage. The higher subsystems depend on some, but not necessarily all of, subsystems below them, without meaning necessarily being attached to the horizontal position of a subsystem relative to the subsystems in lower layers. For example, the WAP stack, Bluetooth®, Infrared and Narrow band are in fact plug-in protocols within the comms infrastructure. Also Messaging depends on the WAP stack, but Java doesn't depend on Infrared. On the other hand, the diagram does attempt where possible to place related subsystems near each other, for example the comms subsystems.



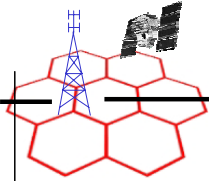
# Symbian OS - Architettura





# Symbian OS - Architettura

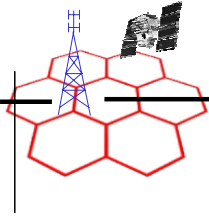
Component	Description
<b>Base</b>	The fundamental runtime system, and low-level security.
<b>Application framework</b>	Middleware APIs for data management, text, clipboard, graphics, internationalization, and core GUI components.
<b>Multimedia</b>	Audio recording and playback, and image related functionality.
<b>Communication infrastructure and network stacks</b>	Wide-area communications stacks including TCP/IP, GSM, GPRS and WAP. Personal-area communications stacks including infrared, Bluetooth® and serial.
<b>Messaging</b>	Internet email, SMS text messages and fax.
<b>Browsing</b>	WML and HTML browsing engines.
<b>Application protocols, services &amp; engines</b>	Engines for contact management, schedule and to-do list management, and other applications.
<b>Java</b>	PersonalJava 3.0 specification JVM-based Java runtime system, with JavaPhone 1.0 APIs.
<b>Connectivity</b>	Converters and viewers for foreign data formats including Microsoft Word email attachments. Communications framework for connecting with PC running Symbian Connect.
<b>Tools</b>	Tools for building applications and ROMs, and for in-target debugging.



# Symbian OS – Kernel & User Library

---

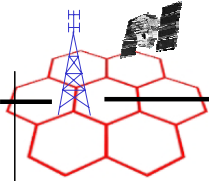
- ❑ The E32 component comprises the kernel **ekern.dll** and user library **euser.dll**;
- ❑ The kernel runs in privileged mode, owns device drivers, does power management, allocates memory to itself and user-mode (that is, unprivileged) processes. It runs natively on **ARM**.
- ❑ The user library offers services to user programs:
  - process, thread, program and memory management
  - error handling and cleanup framework
  - descriptors: strings of characters and buffers of binary data
  - container classes: arrays and lists
  - active objects, for event-driven multi-tasking without requiring the overheads of multi-threading
  - client-server architecture, for simple and efficient inter-process communication
  - a hardware abstraction layer (HAL) presenting a consistent interface to hardware across all devices
  - locale support including currency, time and date formatting
  - miscellaneous services such as timers
  - client-server architecture



# Symbian OS – Application framework

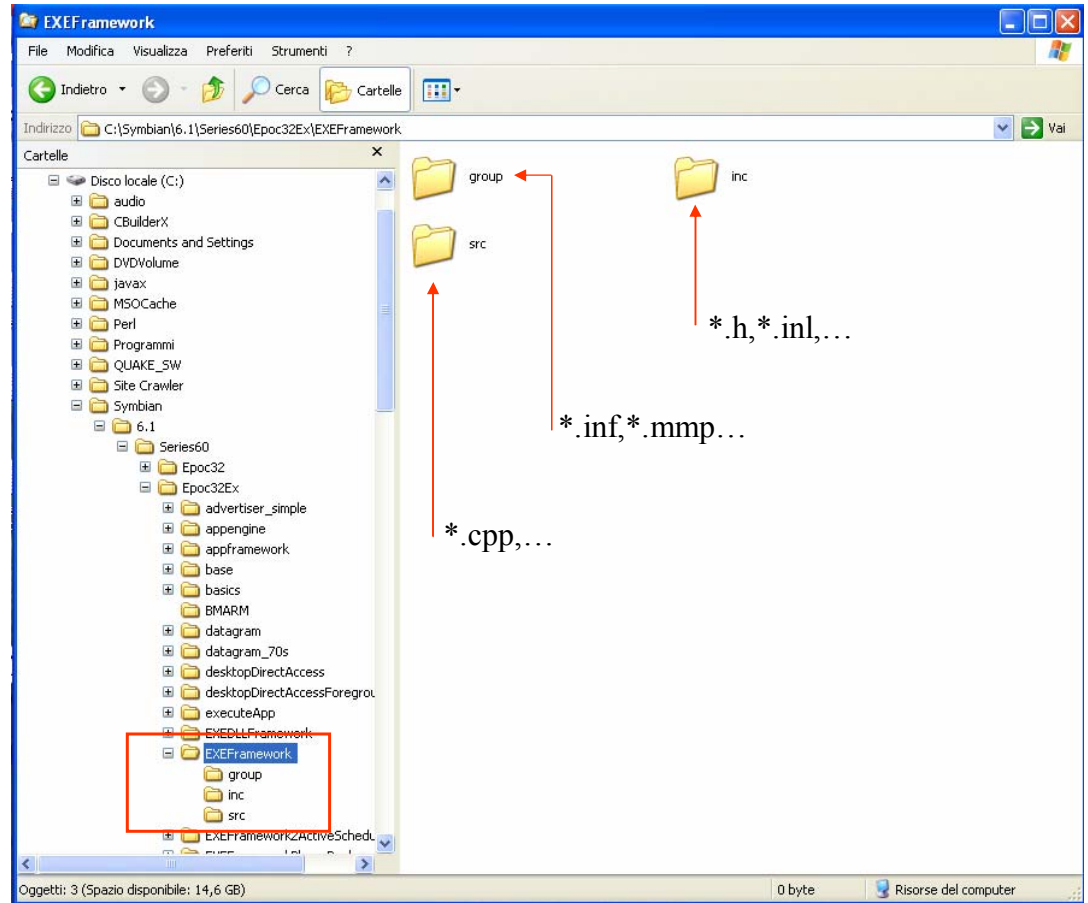
---

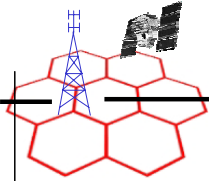
- ❑ Application framework components provide powerful reusable libraries for data, graphics and text, and the lower levels of the application framework;
- ❑ These components include:
  - a relational database manager supporting multiple tables, client-server access for sharing, transactions and rollback, C++ APIs and also SQL for data manipulation and data definition, indexing, windowed query evaluation, 256 columns per table, no practical limits on table size
  - clipboard support
  - graphics for bitmapped devices
  - printing including printer drivers for real printers, and for PC-connected printers via Symbian Connect
  - application architecture, which recognizes files by UID and MIME type, launches and controls applications, and supports object embedding
  - the window server, a windowing system for sharing screen, keyboard and pointer between applications; clocks and animated bitmaps; control framework for sharing an application window between application components
  - rich text, providing a text model with character and paragraph formatting, embedded graphics, and a text view which supports efficient formatting, display and interaction
  - Uikon, the foundational GUI framework
  - internationalization support



# Symbian OS – Esempio: applicazione a console

ExeFramework.exe





# Symbian OS – Esempio: applicazione a console

**bld.inf**

```
File Modifica Formato Visualizza ?
PRJ_MMPFILES
EXEFramework.mmp
```

**ExeFramework.mmp**

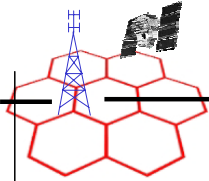
```
File Modifica Formato Visualizza ?
// EXEFramework.mmp

TARGET                EXEFramework.exe
TARGETTYPE            exe
UID                   0

SOURCEPATH            ..\src
SOURCE                EXEFramework.cpp

USERINCLUDE            ..\inc
SYSTEMINCLUDE         \Epoc32\include
SYSTEMINCLUDE         \Epoc32\include\ecom

LIBRARY                euser.lib
```



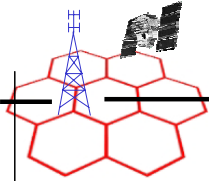
# Symbian OS – Esempio: applicazione a console

---

```
// EXEFramework.H
#ifndef __EUSTD_H
#define __EUSTD_H

#include <e32base.h>
#include <apgtask.h>
#include <eikenv.h>
#include <eikmsg.h>
_LIT(KTtxtEPOC32App,"EXEFramework");
LOCAL_C void startAppL(); // initialize with cleanup stack, then run function code
GLDEF_C TInt E32Main() // main function called by E32
    {
        CTrapCleanup* cleanup=CTrapCleanup::New(); // get clean-up stack
        TRAPD(error,startAppL()); // more initialization, then do app code
        __ASSERT_ALWAYS(!error,User::Panic(KTtxtEPOC32App,error));
        delete cleanup; // destroy clean-up stack
        return 0; // and return
    }
#endif
```

**ExeFramework.h**



# Symbian OS – Esempio: applicazione a console

---

```
#include "EXEFramework.h"
```

**ExeFramework.cpp**

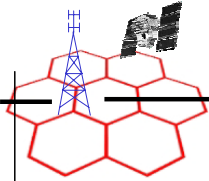
```
LOCAL_C void startAppL()
{
    CConsoleBase* console=Console::NewL(_L("Hello Console"),TSize(KConsFullScreen,KConsFullScreen));
    CleanupStack::PushL(console);

    console->Printf(_L("Press any key...\n"));
    TKeyCode keyCode=console->Getch();

    console->Printf(_L("KEY PRESSED: %c\n"),keyCode);

    console->Getch();

    CleanupStack::PopAndDestroy(); // close and destroy console
}
```



# Symbian OS – Esempio: applicazione a console

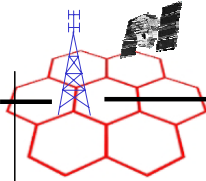
---

**Build project**

```
C:\WINDOWS\system32\cmd.exe
C:\Symbian\6.1\Series60\Epoc32Ex\EXEFramework\group>bldmake bldfiles
```



```
C:\WINDOWS\system32\cmd.exe
C:\Symbian\6.1\Series60\Epoc32Ex\EXEFramework\group>bldmake bldfiles
C:\Symbian\6.1\Series60\Epoc32Ex\EXEFramework\group>abld build wins urel
```



# Symbian OS – Esempio: applicazione a console

**Execute project**

