>>>Guida MIDP 1.0
Il profilo MIDP 1.0 - Mobile Information Device Profile - è la base di j2me, il primo pacchetto di librerie necessarie per sviluppare le prime applicazioni J2ME per i nostri cellulari. In questa guida, indispensabile per chi vuole cominciare a utilizzare questa tescnologia, verrà spiegato quali sono i tool indispensabile per cominciare a programmare, come creare le prime applicazioni e come realizzare il package per distribuire la nostra applicazione.

>>> Corso di base MIDP 1.0

Corso J2ME > Introduzione e profili MIDlet
scritto da Matteo Zinato

J2ME (Java 2 Micro Edition) deriva da Java 2 Standard Edition J2SE, ottimizzato e "leggero" per girare su cellulari e Pda di ultima generazione. Si affianca quindi a J2EE - Java 2 Enterprise Edition, dedicato alle applicazioni server.

Un dispositivo abilitato J2ME consente di scaricare un'applicazione "**MIDlet**" di pochi kbyte (alcuni terminali permettono ad esempio applicazioni di max 10KB ciascuna, e 50 applicazioni totali sul terminale): tali applicazioni vengono installate automaticamente sul device e possono essere utilizzate anche off-line.
Oltre che per le limitazioni di memoria e di potenza di elaborazione, J2ME deve adattarsi anche a display particolarmente ridotti, ad esempio un piccolo telefono cellulare può disporre di 12.288 pixel (96 ×128), oppure un PDA di 20.000, anche se ultimamente l'evoluzione tecnologica in questo senso tende ad abbattere queste limitazioni con display sempre più ampi e con sempre più colori.
Il tutto si basa una Java Virtual Machine, presente nei terminali, che interpreta ed esegue l'applicazione. J2ME comprende due livelli di configurazione (CDC e CLDC) ed inoltre il livello "Profile" MIDP.

L'idea della Sun è quella di partire da una base comune a tutti i dispositivi, con un set di API's limitato a fornire le funzionalità base per un qualunque device a limitata configurazione. Saranno poi i produttori di particolari categorie ad aggiungere funzionalità al CLDC, sotto forma di librerie, generando così quello che è un profilo. Il primo profilo proposto è il **MIDP** (Mobile Information Device Profile), che riguarda principalmente terminali wireless, in grado di instaurare connessione basata sul protocollo HTTP.

Le due configurazioni sono legate a differenti scenari, e comunque sono in forte evoluzione:

Connected Device Configuration (CDC)
Per dispositivi "things that you plug into a wall", cioè collegati in rete, possibilmente always on e ad alta banda, ad es. i palmari
•512 kB minimo di memoria per l'applicazione Java (codice)
•256 kB minimo di memoria "runtime", allocata dall'applicazione
Su tali dispositivi può ad esempio essere incluso PersonalJava.

Connected Limited Device Configuration (CLDC)
Per dispositivi mobili, "things that you hold in your hand", caratterizzati da connettività wireless, ridotta banda e accesso discontinuo
•128 kB di memoria per Java
•32 kB di memoria "runtime"
•interfaccia utente ridotta
•bassi consumi, alimentazione a batteria

Il profilo MIDP - Mobile Information Device Profile - è una estensione, una serie di librerie che aiutano a sviluppare applicazioni tramite API predefinite per interfacciarsi con il terminale (modalità di input, gestione degli eventi, memoria persistente, timer, interfaccia video...). Oltre a MIDP, sullo strato CLDC, si appoggiano anche KJava e EmbeddedJava.
KVM è la **Java Virtual Machine** che viene installata sui terminali, caratterizzata da un ridotto fabbisogno di memoria (40/80 kB + 20/40 kB dinamica) e ridotte potenze di elaborazione (processori 16bit a 25MHz).

Corso J2ME > Vantaggi rispetto alle altre tecnologie
scritto da Matteo Zinato

I vantaggi rispetto a modalità preesistenti come il Wap e Sim Toolkit, risiedono in tre punti principali:

- la possibilità di sfruttare le applicazioni in locale, residenti e funzionanti anche off-line
- la capacità elaborativa
- la flessibilità nell'installazione di nuove applicazioni

Lo sviluppo in Java, inoltre, è facilmente derivabile dallo sviluppo in C++, si possono cioè riconvertire risorse e applicazioni agevolmente. Inoltre l'interfacciamento con le piattaforme web che lavorano con JAVA (Servlet o JSP) è pressocchè immediato. Probabilmente è proprio su questa seconda parte che vi sono più possibilità di business, non tanto quindi sull'applicazione residente sul device.

Sebbene si tratti di uno standard aperto, non mancano alcune peculiarità che possono causare **problemi di compatibilità** tra terminali differenti, proprio per il problema visto nella lezione precedente della creazione dei profili. Siemens, ad esempio, ha definito una serie di librerie proprietarie che risiedono solo sui propri terminali, e che permettono di controllare alcune funzionalità del telefono (ad esempio la vibrazione).
Oltre alle librerie proprietarie legate al terminale, vi è la disponibilità di differenti Virtual Machine (gli interpreti delle applicazioni Java), se ne citano due a titolo di esempio:

- Aplix JBlend, diffuso in Giappone, non solo sui telefonini ma anche su Pda e videocamere (Sony, Sharp, Toshiba, Casio...)
- PersonalJava, legato alla piattaforma Symbian e appoggiato su CDC.

Le capacità multimediali sono attualmente limitate, infatti non è prevista ad esempio la possibilità di riprodurre file audio e, soprattuto per ragioni di sicurezza, permettere ad applicazioni MIDlet J2ME l'accesso alle informazioni sulla alla Sim o sui parametri di configurazione del terminale.

Per far fronte ad applicazioni che richiedono l'appoggio di uno o più database, esistono due possibilità:

1. utilizzare il terminale come semplice interfaccia client - quindi come un **browser** - verso un sistema server remoto dove risiedono i dati. Uno dei vantaggi, in questo caso, è il ridotto impegno nell'aggiornamento del software sul device. Non risulta inoltre critica una evenutale rottura o sostituzione del terminale, per questo può essere adatta a personale ad elevata mobilità sul territorio, come trasportatori o addetti alla riparazioni.
2. disporre di applicazioni - ad esempio su Pda - che lavorano su un **database locale**, sincronizzato con quello remoto. In questo caso sono disponibili versioni ridotte di DB, come l'UltraLight di Sybase o DB tool di IBM, che in pochi KB raccolgono le funzionalità principali. In questo caso è importante che la suite di sviluppo SDK preveda una serie di librerie pronte per l'interfacciamento con tali DB.

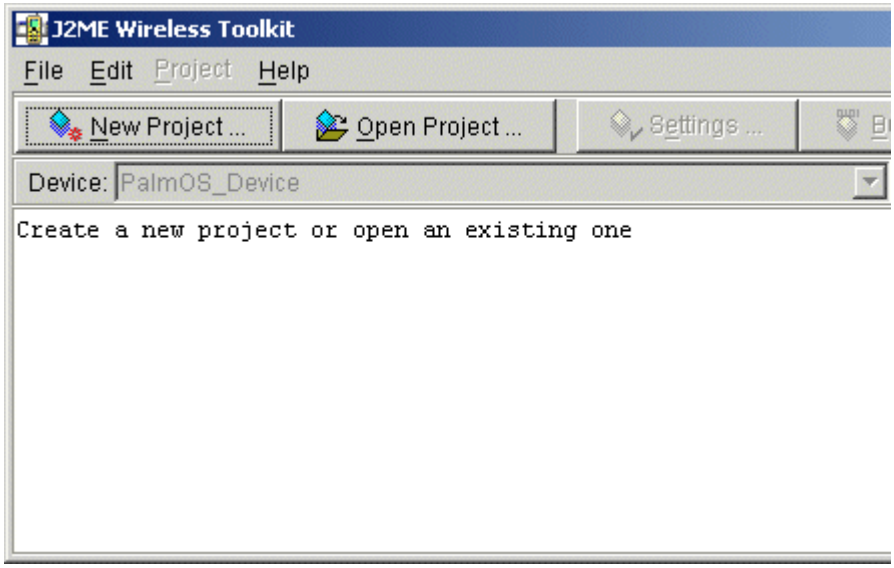
Applicazioni per l'utilità personale e soprattutto giochi sempre più complessi si possono già trovare in internet, in siti specializzati che permettono il download delle applicazioni anche gratuitamente.

Corso J2ME > J2ME Wireless Toolkit: installazione e utilizzo
scritto da Matteo Zinato

Per poter cominciare a creare una prima applicazione J2ME occorre il **J2ME Wireless Toolkit**, messo a disposizione dalla Sun, che consente di compilare e testare con degli emulatori i codice che scriviamo. L'installazione del J2ME Wireless toolkit non richiede nulla di particolare se non un JDK o un JRE 1.3 (o superiore) e la scelta di installare il toolkit integrandolo con Forte for Java, oppure in versione standalone. Per questo corso viene utilizzata la versione standalone che non ha un supporto diretto per la scrittura del codice che deve essere fatta con un editor esterno (per i primi esempi il Blocco Note di Windows va benissimo)

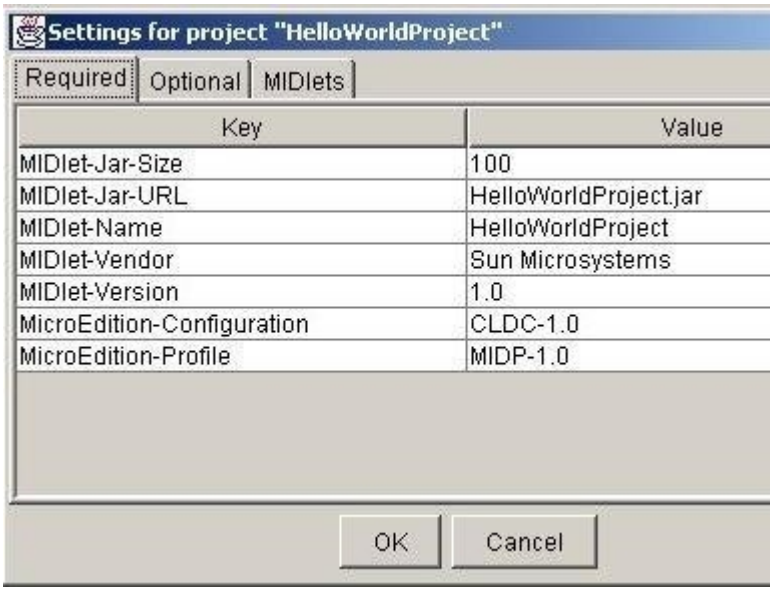
Il Toolkit fornisce una gestione di progetti, ordinandoli seguendo una precisa struttura a cartelle, che deve essere seguita se si vuole realizzare un'applicazione come si deve.

Per creare quindi un progetto apriamo la Ktoolbar, base di partenza per tutte le Midlet che vogliamo creare.



Clicchiamo su *New Project* e specifichiamo due informazioni: nome del progetto (che in verità è il nome della cartella dove viene collocato sul disco) e nome del Midlet iniziale (in futuro potremmo mettere più midlet all'interno dello stesso progetto).

Apparirà poi una schermata dove ci vengono richieste quelle che sono me meta-informazioni sull JAR file da costruire: le ritroveremo poi nel descrittore e nel manifest file. Da notare l'ultimo tab, MIDlets: consente l'aggiunta di ulteriori midlet nello stesso Jar: sarà poi dal terminale che scegliamo quale eseguire in base ad una lista.



Avendo installato con l'opzione standalone, andiamo sulla directory in cui è stato installato J2ME Wireless Toolkit (di solito Wtk104), apriamo la cartella apps che contiene tutti i progetti ed apriamo la cartella HelloWorldProject (o il nome del vostro progetto), che si presenta con questa struttura:

· HelloWorldProject
  > bin
  > res
  > src

all'interno di *bin* dove non si trovano le classi compilate, come sarebbe intuibile, ma il file jar ed il relativo descrittore; *res* è una cartella generica di risorse (spesso immagini PNG) ed *src* è la nostra cartella di lavoro, o meglio la radice del classpath da cui verranno ricercate e compilate le classi dell'applicazione. Alla prima compilazione verranno create le cartelle *classes*, dove questa volta troviamo le singole classi compilate, ed una cartella temporanea.
Il descrittore viene utilizzato per eseguire dei controlli base da parte del software di gestione applicazioni del dispositivo portatile: controllo versioni, dimensioni del jar file, nome autore ecc...

Corso J2ME > Nozioni di base per lo sviluppo
scritto da Matteo Zinato

La scrittura di un MIDlet è abbastanza intuitiva per chi è già pratico di Java e grazie ad uno studio della documentazione, rigorosamente javadoc, e a qualche esempio già scritto, si riesce da subito a realizzare qualcosa di funzionante.
Per la stesura di una MIDlet bisogna però seguire delle regole, né più né meno come si è soliti fare per le applet:

- la classe deve estendere la classe javax.microedition.midlet.MIDlet
- deve implementare i metodi startApp, pauseApp, destryApp (corrispondenti nella funzionalità ai metodi init, stop, destroy delle applet) che sono dichiarati abstract nella classe da cui si estende.

Chi è gia pratico di java noterà, seguendo il corso e leggendo la documentazione, che molte delle classi di largo uso (Vector, Stack…) della J2SE sono rimaste inalterate, con le dovute limitazioni del caso, ma sono state completamente eliminate, per motivi che adesso vediamo, tre utili funzionalità:

- numeri a virgola mobile, dato che non si può essere certi che un CLDC abbia hardware necessario
- JNI (Java Native Interface), per un eccessivo consumo di memoria…
- Introspezione, quindi serializzazione ed RMI non supportati.

Per poter scrivere sul display occorre fare uso delle funzionalità grafiche (che non sono direttamente derivate dalle AWT/Swing) molto semplici data la natura più limitata della GUI di un CLDC. E' stato deciso di seguire il paradigma di contenitori e componenti, come tutta la tecnologia Java: il principale contenitore è il Display Manager implementato dalla classe javax.microedition.lcdui.Display. Ad un Display è associato un oggetto Displayable, che non è altro che un generico contenitore di oggetti grafici che è possibile visualizzare sul display. Quest'ultimo opportunamente esteso classifica due categorie, caratterizzate dalla gestione ad alto o a basso livello della grafica e degli eventi: lo Screen e sottoclassi per il primo caso, il Canvas nel secondo.

Corso J2ME > Hello World: il primo esempio di codice
scritto da Matteo Zinato

Come detto nella lezione dedicata al Wireless Toolkit i nostri codici sorgenti, salvati in file cono estensione .java, vanno posizionati nella cartella X:/wtk104/apps/HelloWorldProject/src. Creiamo il file HelloWorld.java, e scriviamo il codice, iniziando dagli import:

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

Il primo package è costituito soltanto dalla classe MIDlet (astratta) la classe da cui derivare tutte le nostre midlet, il secondo è il package base per la gestione del' interfaccia grafica, che vedremo in un altro articolo.
Dichiaro la classe estendendo MIDlet:

```
public class HelloWorld extends MIDlet {
        Display display;
        Form form;
    .
    .
    .
}
```

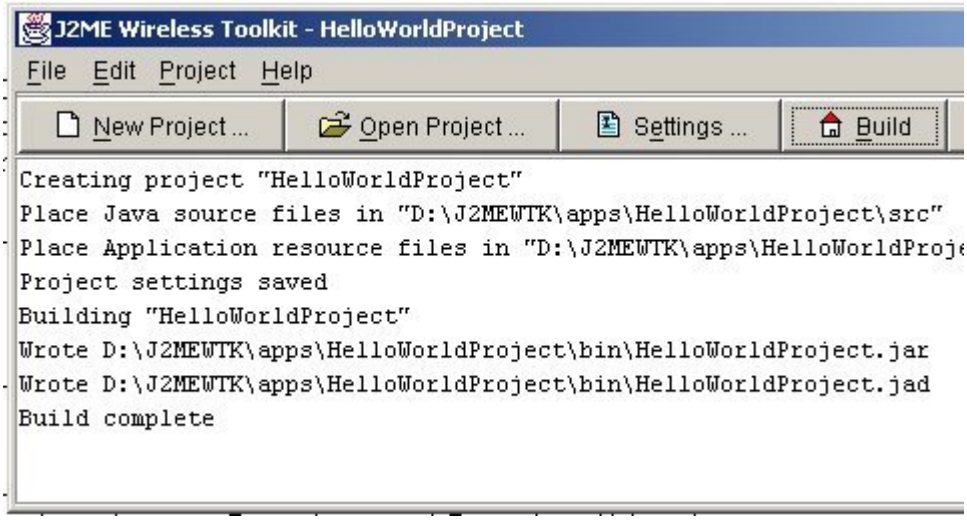
L'attributo display, come detto, è l'astrazione del display del cellulare sul quale impostare qual'è il componente da visualizzare, nel nostro caso il form. Una volta dichiarati gli attributi, passiamo alla definizione dei metodi. Scelta obbligata ricade sui metodi startApp, pauseApp e destroyApp, ma nessuno ci vieta di aggiungerne altri. Per le nostre esigenze è sufficiente implementare lo startApp, invocato una volta istanziato l'oggetto, dato che dovremo gestire un unico evento, quello del lancio dell'applicazione per visualizzare una scritta.

```
public void startApp() {
        display = Display.getDisplay(this);
        //ottengo il display
        form = new Form("WMLScript.it");
        // creo il contenitore
        StringItem sItem = new StringItem(null,"Hello World!");
        //creo il componente
        form.append(sItem);
        //aggiungo il componente al contenitore
        display.setCurrent(form);
        //imposto come displayable corrente
}
```

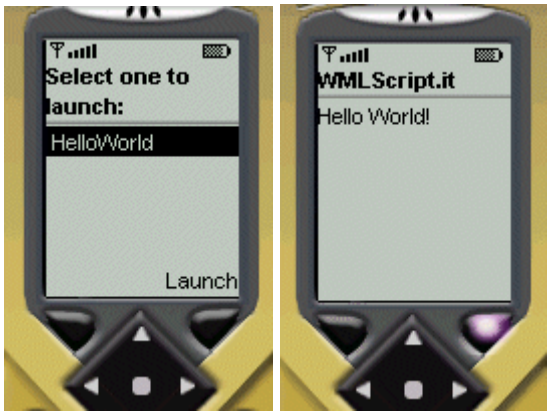
Il sorgente completo quindi dovrebbe apparire così:

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


public class HelloWorld extends MIDlet {
  Display display;
  Form form;

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

  public void pauseApp() {
  }

  public void startApp() {
    display = Display.getDisplay(this);
    form = new Form("WMLScript.it");
    StringItem sItem = new StringItem(null,"Hello World!");
    form.append(sItem);
    display.setCurrent(form);
  }
}
```

Il metodo notifyDestroyed() che viene invocato in destroyApp() è un metodo della superclasse MIDlet, che notifica lo stato Destroyed della MIDlet, quindi la KVM può deallocare tramite Garbage Collection le risorse utilizzate.
Andiamo sul Ktoolbar del WK, clicchiamo su Build (che non fa altro che compilare i file java) e dovrebbe dare come risultato la seguente schermata:



In questo caso non ci sono stati errori di complilazione e la nostra mildet puo essere eseguita con il pulsante **run** che caricherà l'applicazione su uno degli emulatori selezionati
Il risultato sarà il seguente:



La prima schermata che appare sull'emulatore è di fatto la lista delle MIDlet presenti nel progetto, la seconda , che appare dopo aver selezionato la midlet, è l'esecuzione vera e propria della midlet HelloWorld.

Corso J2ME > I Package J2ME
scritto da Matteo Zinato

Chi ha programmato in java non puo non conoscere il termine package, che non sono altro che delle librerie di classi organizzate in modo gerarchico. Per esempio il package java ha dei sotto package come java.io, java.lang, java.util che contengono le classi necessarie per lo sviluppo delle applicazioni.
Tutta la documentazione tecnica delle classi è fornita dalla sun sottoforma di javadocs (documenti html) e per la spiegazione delle classi contenute nei vari package utilizzeremo proprio questa documentazione ufficiale.

Cominciamo quindi a vedere il contenuto dei vari package:

- **javax.microedition.lcdui**: è il cosi detto User Interface Package che contiene le classi per creare delle interfacce utente sul display.
- **javax.microedition.rms**: utilizzato per la memorizzazione persistente, fornisce dei meccanismi che permettono di memorizzare e recuperare dati in modo persistente.
- **javax.microedition.midlet**: definisce le interfacce i il ciclo di vita delle MIDlet.
- **javax.microedition.io**: detto anche il package del Networking include dei supporti per la connessione alla rete da un Connected Limited Device Configuration. Contiene molte interfacce ma una sola classe Connector utilizzata per creare gli oggetti connection.
- **java.io** si occupa di gestire l'input e l'output da un generico stream di dati
- **java.lang** include tutte le classi tipiche del linguaggio Java incluse nella Java 2 Standard Edition. In questo package troviamo tutte le classi per i tipi di dato (Boolean, Byte, Character, Integer, Long, Short, String, StringBuffer) oltre alle classi basi di sistema e per i Thread (Math, Object, Runtime, System, Thread, Throwable).
- **java.util** include tutte le classi di utility incluse nella Java 2 Standard Edition. In questo package troviamo le più classi di utilità che un programmatore java è abituato ad utlizzare (Calendar, Date, Hashtable, Random, Stack, Timer, TimerTask, TimeZone , Vector)

Corso J2ME > Il Package javax.microedition.lcdui

scritto da Matteo Zinato

In questa pagina sono presentate tutte le classi di questo pacchetto, in ordine alfabetico con una breve descrizione del compito che svolgono. In questo corso vediamo in dettaglio solo questo package per che è quello che contiene il maggior numero di classi, e soprattutto quelle fondamentali per realizzare applicazioni di qualsiasi tipo. Per chi poi volesse approfondire i metodi e gli attributi della singola classe è possibile far riferimento, tramite un link sul nome della classe, alla documentazione tecnica JAVADOCS che la sun ha rilasciato per gli sviluppatori.

Come detto il package javax.microedition.lcdui (User Interface Package) contiene le classi per creare delle interfacce utente sul display. Con le classi appartenenti a questo package è infatti possibile gestire dei form per l'inserimento dei dati con campi di testo, selezioni a scelta multipla, date o ore in formato grafico, immagini e così via.
In seguito vedremo un esempio di codice che utilizza alcuni degli oggetti più significativi presentati in questa pagina.

Alert	Un Alert non è altro che una schermata che mostra delle informazioni all'utente per un certo periodo di tempo per poi procedere con la schermata successiva. Viene utilizzato per non cambiare il contenitore grafico che si sta utilizzando.
AlertType	The AlertType indica il tipo di alert che si sta utilizzando e il suo utilizzo più comune è la generazione di suoni da un alert.
Canvas	E' una classe utilizzata per le applicazioni che hanno la necessità di intercettare le azioni dell'utente sui tasti del dispositivo (e di un eventuale puntatore se previsto). Permette di abbinare le azioni sulla tastiera a determinati eventi grafici sul display.
ChoiceGroup	Un ChoiceGroup è un gruppo di elementi che possono essere selezionati dall'utente in modo che l'applicazione agisca di conseguenza.
Command	E' una classe che racchiude le informazioni di un comando. Da notare che non si specifica con questa classe cosa deve fare l'applicazione in seguito a un determinato comando (questo compito è del CommandListner), ma solo gli attributi del comando stesso.
DateField	Un DataField è un campo data da inserire in un form per l'inserimento dei dati, che permette di inserire e visualizzare la data in formato grafico con una sorta di calendario
Display	Rappresenta l'oggetto che di fatto gestisce il display del dispositivo e i sistemi per l'inserimento dei dati (la tastiera ad esempio tastiera). Ha metodi per ottenere le caratteristiche del terminale e per conoscere quali oggetti possono essere visualizzati.
Displayable	E' u generico oggetto grafico che può essere visualizzato sul display.
Font	Rappresenta i tipici font che siamo abituati ad utilizzare.
Form	E' un oggetto che contiene un insieme di altri oggetti come campi per il testo, immagini, radio botton, campi per le date ecc...
Gauge	E' una sorta di grafico a barre che può essere utilizzato in un form per l'inserimento di una grandezza.
Graphics	E' una classe che contine dei semplici metodi per il disegno di figure geometriche

	in 2 dimensioni
Image	E' un oggetto che serve per gestire la visualizzazione delle immagini
ImageItem	E' una classe che permette di avere un controllo del posizionamento di un'immagine quando questa è inserita in un form o in un alert
Item	E' una superclasse per tutti quei componenti che possono essere aggiunti ad un form o ad un alert.
List	Rappresenta una schermata che contiene una lista di scelte per l'utente.
Screen	E' una classe per l'utilizzo di interfacce utente ad alto livello. Permette infatti di visualizzare dei titoli opzionali o delle scritte scorrevoli sull'oggetto Display.
StringItem	Non è altro che un item che può contenere delle stringhe
TextBox	E' un oggetto che permette all'utente di inserire del testo ed eventualmente di modificarlo
TextField	E' un oggetto che permette all'utente di inserire del testo in un form. Contiene anche delle proprietà che permettono il controllo automatico dei dati inseriti.
Ticker	Di fatto non è altro che una stringa di testo che scorre sulla parte alta del display.

Corso J2ME > Creare Pulsanti e Menu
scritto da Matteo Zinato

Una delle cose più importanti da conoscere per iniziare a realizzare delle applicazioni è come riuscire a creare dei pulsanti per poter comandare con la tastiera del terminale l'applicazione che realizziamo.
Come detto nella lezione precedente per fare ciò si utilizza l'oggetto Commmand contenuto nel package javax.microedition.lcdui. Vediamo subito un esempio per cominciare a capirne qualcosa:

Creiamo una semplice MIDlet che chiamiamo PulsantiEMenu che come al solito estende la classe MIDlet.

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

public class PulsantiEMenu extends MIDlet{
```

Creiamo gli oggetti di base per il funzionamento della MIDlet: un Display e un Form che sarà il contenitore dei comandi:

```
private Display display;
private Form fmMain;
```

Adesso dichiariamo in nostri oggetti Command. Ad ogni oggetto dichiarato corrisponde un pulsante on un Item di un eventuale menu. E' consigliabile in questa fase dare alle variabili un nome che ne ricordi il funzionamento perchè è facile poi fare confusione.

```
private Command cmExit;
private Command cmPrimo;
private Command cmSecondo;
private Command cmTerzo;
```

Dopo aver dichiarato tutte le varibili, cominciamo con i metodi della MIDlet, costruttore, pauseApp, destroyApp che rimangono vuoti e startApp che verrà invece implementato:

```
public PulsantiEMenu(){
}

public void pauseApp(){
}

public void destroyApp(boolean unconditional){
}

public void startApp(){

    display = Display.getDisplay(this);

    cmExit = new Command("Exit", Command.EXIT, 1);
    cmTerzo = new Command("Terzo", Command.SCREEN, 4);
    cmPrimo = new Command("Primo", Command.SCREEN, 2);
    cmSecondo = new Command("Secondo", Command.SCREEN, 3);
```

A questo punto è necessario dare qualche spiegazione: ogni variabile di tipo Command che avevo dichiarato sopra viene inizializzata con tre parametri propri dell'oggetto Command:

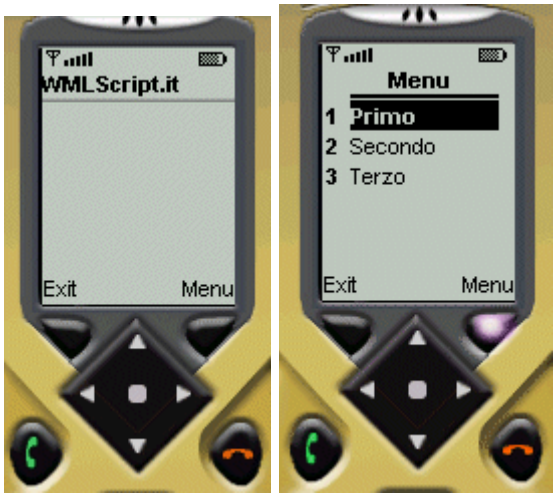
- **Un etichetta di testo**, che rappresenta cosa viene effettivamente visualizzato sul display per quel comando
- **Un tipo**, che specifica a cosa serve esattamente quel comando. Specificando il tipo di un comando, il dispositivo che interpreta l'applicazione posizione quel comando in modo standard per tutte le applicazioni (per esempio il tasto BACK sempre in basso a destra). I possibili tipi sono BACK, CANCEL, EXIT, HELP, ITEM, OK, SCREEN, e STOP.
- **Una Priorità**, che descrive l'importanza del comando relativamente agli altri comandi presenti sullo schermo (più basso è il numero più il comando è importante). Anche in questo caso la scelta del posizionamento dei comandi viene fatta dal dispositivo che interpreta l'applicazione. In base alla priorità e al tipo del comando la JVM del dispositivo deciderà se visualizzare i comandi tutti sulla stessa schermata, organizzarli in menu o altro, anche a seconda dei tasti messi a disposizione.

A questo punto continuiamo a vedere il codice. Ora è necessario aggiungere tutti i comandi all'oggetto Form che in questo caso è il nostro contenitore.

```
    fmMain = new Form("WMLScript.it");

    fmMain.addCommand(cmExit);
    fmMain.addCommand(cmPrimo);
    fmMain.addCommand(cmSecondo);
    fmMain.addCommand(cmTerzo);

    display.setCurrent(fmMain);
}
```

Con l'ultimo comando viene impostato il form sul display.

Ora vediamo qual'è il risultato sull'emulatore del codice. Sull'emulatore standard del Wireless Toolkit il risultato è il seguente



L'applicazione viene invece interpretata in modo diverso da un altro tipo di terminale, proprio per il discorso fatto prima (infatti l'emulatore della motorola ha un tasto menu che cambia un pò le carte in gioco)

Corso J2ME > Intercettare gli eventi
scritto da Matteo Zinato

I più attenti di voi avranno notato che nell'esempio visto nella lezione precedente abbiamo creato i nostri pulsanti o menu ma non abbiamo associato a questi nessun tipo di azione ai comandi, chi ha provato ad eseguire il codice avrà notato che premendo i vari pulsanti non accadeva nulla.
Per fare ciò di cui abbiamo bisogno riprendiamo l'esempio della lezione sui bottoni e sui menu e aggiungiamo quello che ci serve per associare ai pulsanti delle determinate azioni.

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

public class PulsantiEventi extends MIDlet implements CommandListener{
```

Come al solito la nostra MIDlet PulsantiEventi estende la classe MIDlet, ma oltre a fare ciò implementa CommandListener. CommandListener è un'interfaccia del package javax.microedition.lcdui che ha un solo metodo commandAction che deve essere obbligatoriamente riscritto. Creiamo gli oggetti di base per il funzionamento della MIDlet: un Display, un Form e i in nostri oggetti Command.

```
private Display display;
private Form fmMain;

private Command cmExit;
private Command cmPrimo;
private Command cmSecondo;
private Command cmTerzo;
```

In coda a questi oggetti ne aggiungiamo poi altri due, rispettivamente una textBox e un altro comando che gestirà l'uscita dalla schermata con la textBox. L'intento è quello di associare un evento dl tipo "visualizza la textBox" ai comandi cmPrimo, cmSecondo e cmTerzo.

```
private TextBox tbAction;
private Command cmBack;
```

Dopo i soliti metodi obbligatori cominciamo ad implementare startApp

```
public PulsantiEMenu(){
}

public void pauseApp(){
}

public void destroyApp(boolean unconditional){
}

public void startApp(){

    display = Display.getDisplay(this);

    cmExit = new Command("Exit", Command.EXIT, 1);
    cmBack = new Command("Back", Command.BACK, 1);
    cmTerzo = new Command("Terzo", Command.SCREEN, 4);
    cmPrimo = new Command("Primo", Command.SCREEN, 2);
    cmSecondo = new Command("Secondo", Command.SCREEN, 3);
```

Creiamo l'istanza dei comandi, ricordandondoci stavolta anche di cmBack

```
    fmMain = new Form("WMLScript.it");
    fmMain.addCommand(cmExit);

    fmMain.addCommand(cmPrimo);
    fmMain.addCommand(cmSecondo);
    fmMain.addCommand(cmTerzo);
    fmMain.setCommandListener(this);
```

Aggiungiamo i nostri comandi (ma non il Back che va in un'altra schermata) al nostro form, settando poi il CommandListener sul form. In questo modo diciamo all'applicazione di stare in un certo senso in ascolto di quello che succede.

Procediamo creando l'istanza della nostra textBox. La textBox rappresenta in questo caso una schermata indipendente e quindi ci fa da contenitore, come fa il form per la visualizzazione del menu. Aggiungiamo quindi il comando cmBack e associamo il CommandListener anche alla textBox.

```
    tbAction = new TextBox("TextBox", "Inserisci i tuoi dati", 25, 0);
    tbAction.addCommand(cmBack);
    tbAction.setCommandListener(this);
```

Per concludere il metodo startApp() impostiamo il display sul Form, è infatti quello che vogliamo vedere inizialmente.

```
    display.setCurrent(fmMain);
}// fine startApp()
```

A questo punto viene il bello: abbiamo creato i nostri pulsanti, abbiamo detto all'applicazione di stare in ascolto sul form e sulla textBox, ora dobbiamo dire alla nostra MIDlet cosa deve fare quando premiamo un determinato pulsante.
Questo è fatto com il metodo commandAction dell'interfaccia commmandListener. Il commandAction ha due parametri, il primo di tipo Command rappresenta il pulsante che è stato premuto, il secondo, di tipo Displayable, rappresenta invece il contenitore a cui appartiene il comando.

Se per esempio viene premuto il pulsante cmPrimo, su commandAction si avrà che c = cmPrimo e s = fmMain (il form a cui appartiene il comando cmPrimo).

```
public void commandAction(Command c, Displayable s){
    if (c == cmExit){
        destroyApp(false);
        notifyDestroyed();
    }else if (c == cmTerzo || c == cmSecondo || c == cmTerzo)
        display.setCurrent(tbAction);
    else if (c == cmBack)
        display.setCurrent(fmMain);
}
```

Come avrete notato, quando viene premuto Exit viene chiusa l'applicazione, se viene premuto uno degli altri tre pulsanti del menu il display viene settato su tbAction (la textBox). Infine se si preme il Back presente nella texBox si torna a fmMain (il form).

Corso J2ME > Utilizzare gli oggetti per l'inserimento dei dati
scritto da Matteo Zinato

Nel package javax.microedition.lcdui sono presenti molti oggetti che vengono utilizzati per l'inserimento dei dati in un form da parte di un utente. Oltre alle tipiche textBox, textArea, radioBotton e checkBox sono presenti anche oggetti DateField e Gauge che servono per l'inserimento visuale di date, ore e unità di grandezza per via di visualizzazioni grafiche.

Vediamo subito un esempio di codice che utilizza tutti questi oggetti:

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

public class FormDemo extends MIDlet{
```

Dopo aver importato i package e scritto l'intestazione della classe, dichiariamo al solito un oggetto Display e un oggetto Form che sarà il contenitore di tutti gli altri oggetti dichiarati di seguito.

```
    Display d;
    Form aForm;

    ChoiceGroup aChoiceGroup1;
    ChoiceGroup aChoiceGroup2;
    DateField aDateField;
    DateField aTimeField;
    Gauge aGauge;
    StringItem aStringItem;
    TextField aTextField;
```

A questo punto creiamo il costruttore che al solito crea le istranze delle classi.

```
public FormDemo () {
    aForm = new Form("WMLScript.it");

    // una semplice stringa per la visualizzazione di un titolo
    aStringItem = new StringItem(null,"Compila i seguente form");
```

Per prima cosa vediamo come creare delle opzioni di scelta multipla. Con lo stesso oggetto ChoiceGroup è possibile creare sia dei radioBotton (in cui può essere selezionato solo un elemento), sia delle checkBox (in cui possono essere selezionati n elementi). Nella documentazione appare anche un terzo tipo (IMPLICIT) che però, proprio la documentazione indica come non utilizzabile con l'oggetto ChoiceGroup.

Il vettore di stringe choices[] rappresenta le stringhe delle singole scelte.

```
    String choices[] = {"Primo", "Secondo"};

    // stampa le liste di scelta multipla
    aChoiceGroup1 = new ChoiceGroup("Scegli",
                                    Choice.EXCLUSIVE,
                                    choices,
                                    null);


    aChoiceGroup2 = new ChoiceGroup("Scegli3",
                                    Choice.MULTIPLE,
                                    choices,
```

```
null);
```

Ecco il risultato di questa prima parte:

Vediamo ora l'oggetto DateField che permette al programmatore di creare delle interfacce grafiche per l'inserimento di date e ore. Con una semplice riga di codice è possibile fare in modo che all'utente venga presentato un calendario o un orologio da cui scegliere il valore di data o ora.

```
aTimeField = new DateField(null,DateField.TIME);

aDateField = new DateField(null,DateField.DATE);
```

Con il seguente risultato:

La classe Gauge permette invece di visualizza un grafico su cui si può impostare il numero di colonne piene. L'oggetto viene istanziato passando come parametri:

- Una stringa con il titolo
- Una valore buleano che indica se l'utente può cambiarne il valore
- Un intero che rappresenta il valore massimo del grafico
- Un intero che rappresenta il valore iniziale

```
aGauge = new Gauge("Punteggio",true, 10, 1);
```

Con il seguente risultato:

Infine creiamo un'stanza dell'oggetto TextField, che permette all'utente di inserire del testo in un form e che contiene anche delle proprietà che permettono il controllo automatico dei dati inseriti. L'oggetto viene istanziato passando come parametri:

- Una stringa con il titolo
- Una stringa che rappresenta il contenuto iniziale della textBox
- Un intero che rappresenta il numero massiomo di caratteri che possono essere inseriti
- Una costante che rappresenta il filto da applicare all'inserimento:

  TextField.**ANY**: ammessi tutti i caratteri
  TextField.**EMAILADDR**: la stringa inserita deve essere un indirizzo e-mail
  TextField.**NUMERIC**: solo caratteri numerici
  TextField.**PASSWORD**: la stringa inserita viene "coperta" con degli *
  TextField.**PHONENUMBER**: la stringa inserita deve essere un numero di telefono sintatticamente corretto (non si sà con che criterio) in modo chè talune applicazioni possano avviare direttamente una chiamate
  TextField.**URL**: la stringa inserita deve essere un indirizzo internet

```
    aTextField = new TextField("Commenti",
                               "Scrivi qui",
                               20,
                               TextField.ANY);
```

Per concludere, come ormai avrete capito, "appendiamo" tutti i nostri oggetti al nostro contenitore Form:

```
    aForm.append(aStringItem);
    aForm.append(aChoiceGroup1);
    aForm.append(aChoiceGroup2);
    aForm.append(aDateField);
    aForm.append(aTimeField);
    aForm.append(aGauge);
    aForm.append(aTextField);
}
```

e per completezza chiudiamo la classe con gli altri metodi richiesti:

```
protected void startApp() {
    d = Display.getDisplay(this);
    d.setCurrent(aForm);
}

protected void pauseApp() {
}

protected void destroyApp(boolean unconditional) {
}

}// fine classe FormDemo
```

Corso J2ME > Utilizzare gli oggetti per la grafica
scritto da Matteo Zinato

Fino ad ora nel nostro corso abbiamo visto come realizzare delle semplici applicazioni che visualizzavano su display degli oggetti piuttosto standard come textBox, radiobotton o al massimo qualche grafico non personalizzabile. In questa lezione vediamo invece come realizzare della grafica in modo personale attraverso la classe Canvas.

L'oggetto Canvas, come detto nella pagina con il riepilogo delle classi del package javax.microedition.lcdui è utilizzato per visualizzare della grafica sul display e per le applicazioni che hanno la necessità di intercettare le azioni dell'utente sui tasti del dispositivo (e di un eventuale puntatore se previsto). In questa pagina ci occuperemo solo della grafica lasciando gli eventi sui pulsanti alla prossima lezione.

I metodi che ci interessano sono dunque pochi e servono per conoscere le dimensioni dell'are su cui si può disegnare (getHeight() e getWidth()) e per disegnare (paint()) tramite l'oggetto Graphics che contiene tutti i metodi per scrivere testo, visualizzare figure geometriche, impostare i colori ecc.
In pratica ogni applicazione che dovrà utilizzare delle interfacce grafiche dovrà essere formata da due classi:

- una classe che estende Canvas e implementa il metodo paint() su cui andremo a scrivere tutto il codice per la grafica
- una seconda classe che come al solito estende MIDlet che non farà altro che creare un'istanza della classe appena vista.

In alcuni casi è possibile evitare di creare due classi, implementando tutto all'interno della MIDlet principale, ma è sempre bene mantenere separate le due cose.

La classe che estende MIDlet e che non fa altro che creare un'istanza di un'altra classe che gestisce la grafica è molto semplice e ne vediamo qui un esempio:

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

public class CanvasMIDlet extends MIDlet{

  private Display  display;
  private MyCanvas canvas;

  public CanvasMIDlet()
  {
    display = Display.getDisplay(this);
    // crea l'stenza della nostra classe che estende canvas
    canvas  = new MyCanvas();
  }

  protected void startApp()
  {
    // assegna al display canvas,
        // in modo da visualizzarne il contenuto
    display.setCurrent(canvas);
  }

  protected void pauseApp()
  { }
```

```
  protected void destroyApp( boolean unconditional )
  { }

  public void exitMIDlet(){
    destroyApp(true);
    notifyDestroyed();
  }
}
```

Passiamo ora a vedere l'altra classe, MyCanvas.java, in cui utilizzeremo gli oggetti grafici. Come detto sopra la cosa da fare è piuttosto semplice, si tratta solo di sovrascrivere il metodo paint(), al cui interno andiamo a realizzare la grafica. Nel seguente codice abbiamo utilizzato solo una piccola parte dei tanti metodi a disposizione, ma la cosa importante a questo punto è imparare a conoscerne la logica di funzionamento. Una volta all'interno del metodo **paint()**, l'oggetto **Graphics** g ci fornisce metodi per disegnare rettangoli, arhi, scrivere testo, impostare il colore degli elementi, conoscere le dimensioni del dispay e i colori disponibili ecc.

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

class MyCanvas extends Canvas{

  public TextCanvas()
  {
  }

  protected void paint(Graphics g){
    // pulisce il display
    g.setColor(255, 255, 255);
    g.fillRect(0, 0, getWidth(), getHeight());
```

La prima cosa da fare è pulire il display perchè altrimenti andremmo a disegnare sopra quello che già c'è. E' molto semplice, basta disegnare un rettangolo del colore che vogliamo grande come tutto lo schermo.

```
    // scrive sul display una stringa al centro del display
    g.setColor(0, 0, 0);
    g.drawString("WMLScript.it",
                 getWidth()/2,
                 getHeight()/2,
                 Graphics.TOP | Graphics.HCENTER);
```

Nella scrittura del testo, oltre ai parametri riguardanti le coordinate, è necessario specificare un parametro detto punto di ancora (l'ultimo), che serve per ridurre la quantità di calcoli che il dispositivo deve effettuare per il rendering del testo sul display. Le ancore definite nell'oggetto Graphics sono LEFT, HCENTER e RIGHT per il posizionamento orizzonatale e TOP, BASELINE e BOTTOM per quello vericale. I due tipi di posizionamento possono essere combianati con l'operatore OR.
Molti di voi si chiederanno: "ma non bastava specificare le coordinate?". Il fatto è che per posizinare il testo il dispositivo deve tener conto di un gran numero di variabili (l'altezza e la lunghezza del font, gli altri oggetti presenti ecc). Vediamo subito infatti come posisizioniamo una stringa sotto a quella appena scritta solo modificando un'ancora.

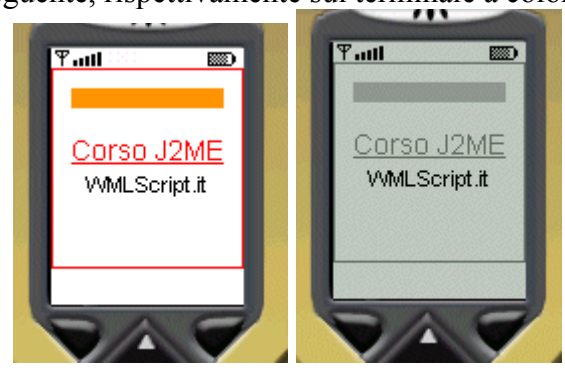
```
    // disegna un rettangolo rosso che fa da contorno al display
    g.setColor(255, 0, 0);
    g.drawRect(0, 0, getWidth()-1, getHeight()-1);
```

```
    // scrive una stringa settando il font da utilizzare
    Font matteo = Font.getFont(Font.FACE_PROPORTIONAL,
                               Font.STYLE_UNDERLINED,
                               Font.SIZE_LARGE);
    g.setFont(matteo);
    g.drawString("Corso J2ME",
                 getWidth()/2,
                 getHeight()/2,
                 Graphics.BOTTOM | Graphics.HCENTER);
```

Come abbiamo appena visto è possibile scrivere delle stringhe personalizzando il font, proprio grazie alla classe Font che ci permette di settare, tramite delle costanti, il tipo di font, lo stile e la grandezza.

```
    // disegan un rettangolo pieno
    g.setColor(0, 255, 255);
    g.fillRect(10, 10, getWidth()-20, 10);

  }
}
```

Il risultato del codice è il seguente, rispettivamente sul terminale a colori e bianco e nero:



Il consiglio, quando si sviluppano applicazioni che implicano l'utilizza di grafica, è quello di cercare il più possibile di parametrizzare il posizionamento, evitando coordinate assolute e utilizzando sempre i metodi getWidth() e getHeight(), derivando da questi la posizione desiderata. In questo modo si garantisce una buona visibilità su tutti i terminali in proporzione alle dimensioni dello schermo.

Corso J2ME > Intercettare le azioni sui tasti
scritto da Matteo Zinato

La classe Canvas, oltre ad essere utilizzata per la grafica, possiede dei metodi che permettono di intercettare le azioni dell'utente sui tasti del dispositivo. Possiamo di fatto capire quali sono i pulsanti che vengono premuti ed associare a questi determinate azioni.
Grafica e pulsanti sono stati uniti nella classe canvas in un modo che potrebbe a prima vista creare confusione, ma con il tempo ci si rende conto che le due funzionalità sono molto legate tra loro (quasi sempre alla pressione di un pulsante si vuole fare corrispondere un evento grafico) e una gestione con la stessa classe risulta molto utile.

Come già visto, ogni applicazione che utilizza la classe canvas è composta da due classi:

- una classe che estende Canvas e implementa i metodi keyPressed() e keyReleased() e associa ad ogni tasto un'azione
- una seconda classe che come al solito estende MIDlet che non farà altro che creare un'istanza della classe appena vista.

La classe che estende MIDlet e che non fa altro che creare un'istanza di un'altra classe che estende Canvas, e questa volta creiamo anche un pulsante per uscire.

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

public class CanvasMIDlet extends MIDlet implements CommandListener{

  private Display  display;
  private MyCanvas canvas;
  private Command exit;

  public CanvasMIDlet(){
    display = Display.getDisplay(this);
        canvas  = new MyCanvas();
  }

  protected void startApp(){
    display.setCurrent(canvas);
    exit = new Command("Exit", Command.STOP, 1);
    canvas.addCommand(exit);
    canvas.setCommandListener(this);
  }

  protected void pauseApp()
  { }

  protected void destroyApp( boolean unconditional )
  { }

  public void exitMIDlet()
  {
    destroyApp(true);
    notifyDestroyed();
  }

  public void commandAction(Command c, Displayable s){
    if(c == exit) {
      notifyDestroyed();
    }
```

```
  }

}//fine classe
```

Passiamo ora a vedere l'altra classe, MyCanvas.java. Come detto sopra il nostro compito è quello di intercettare le pressioni dell'utente sui pulsanti e lo facciamo attraverso i metodi **keyPressed()** e **keyReleased()** che ci segnalano rispettivamente quando viene premuto e quando viene rilasciato un determinato pulsante. Questi due metodi hanno come parametro un intero (keyCode), che corrisponde al codice del pulsante premuto. Un altro metodo molto importante **getGameAction()** permette di ottenere dal pulsante un codice che ha un preciso significato per l'applicazione che stiamo realizzando.

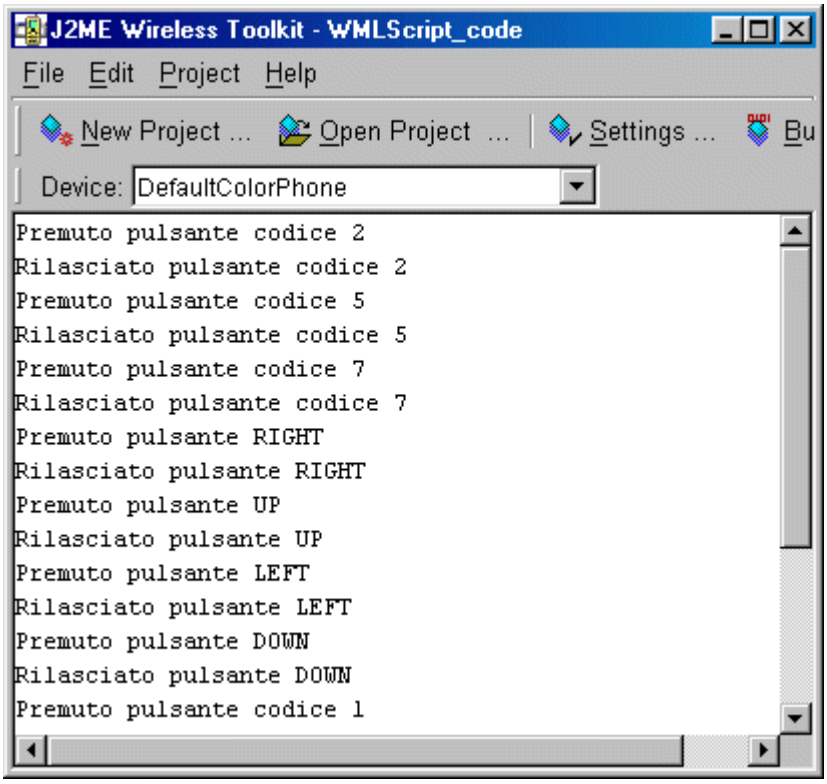
Quando si realizza un'applicazione, pensiamo ad un semplice gioco, che utilizza le quattro frecce e pochi altri pulsanti, è fortemente consigliato utilizzare il metodo getGameAction() piuttosto che mappare i comandi per codice o per nome. I gameAction definiti sono: **UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C, e GAME_D** e l'applicazione può tradurre un codice di un comando in un gameAction con il metodo getGameAction(int keyCode), ed effettuare l'operazione inversa con getKeyCode(int gameAction). Tutta questa procedura, che sembra macchinosa e complessa, è stata creata pensando alla portabilità delle applicazioni. Se scriviamo una MIDlet che necessita delle quattro frecce e utilizziamo il getGameAction(), quell'applicazione potrà essere utilizzata anche da un dispositivo che non possiede quei pulsanti ma che utilizza per quelle funzioni, i pulsanti 2, 4, 6, 8.

Vediamo l'esempio:

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

class MyCanvas extends Canvas {

  public MyCanvas()
  { }

  public void paint(Graphics g)
  { }

  protected void keyPressed(int keyCode) {
    if (keyCode > 0) {
      System.out.println("Premuto pulsante codice "
                         +((char)keyCode));
    } else {
      if (UP == getGameAction(keyCode)){
        System.out.println("Premuto pulsante UP");
      }else if (DOWN == getGameAction(keyCode)){
        System.out.println("Premuto pulsante DOWN");
      }else if (LEFT == getGameAction(keyCode)){
        System.out.println("Premuto pulsante LEFT");
      }else if (RIGHT == getGameAction(keyCode)){
        System.out.println("Premuto pulsante RIGHT");
      }
    }
  }//end keyPressed


  protected void keyReleased(int keyCode) {
    if (keyCode > 0) {
      System.out.println("Rilasciato pulsante codice "
```

```
                          +((char)keyCode));
    } else {
      if (UP == getGameAction(keyCode)){
        System.out.println("Rilasciato pulsante UP");
      }else if (DOWN == getGameAction(keyCode)){
        System.out.println("Rilasciato pulsante DOWN");
      }else if (LEFT == getGameAction(keyCode)){
        System.out.println("Rilasciato pulsante LEFT");
      }else if (RIGHT == getGameAction(keyCode)){
        System.out.println("Rilasciato pulsante RIGHT");
      }
    }
  }//end keyreleased

}// fine classe
```

Nel nostro esempio non abbiamo associato nessuna azione particolare ai nostri pulsanti se non la scrittura sulla console di un messaggio che ci indica quale pulsante è stato premuto o rilasciato. Il risultato per questa volta è visibile quindi sulla toolbar del Wireless Toolkit:



E' facile intuire a questo punto che se invece di una semplice stampa avessimo scritto dei metodi per scrivere della grafica sul dispay nulla sarebbe cambiato, ma lo vedremo nelle prossime lezioni.

Corso J2ME > Leggere un file contenuto nelle risorse
scritto da Matteo Zinato

Nelle prime lezioni di questa quida, nel parlare dell'organizzazione delle directory di un'applicazione J2ME, abbiamo visto che esiste una cartella **res** che contiene dei file che possono essre letti da una midlet. Fino ad ora questi file si sono limitati a qualche immagine che venica caricata con le opportune classi specificando solo il nome del file, ma la cartella res può contenere qualsiasi tipo di file, con qualsiasi nome e qualsiasi estensione, che può essere interpretato dalla nostra applicazione come u semplice stream di dati.

Cominciamo creando un file chiamato "file.test" all'interno della cartella res della notra applicazione. All'interno del file scriviamo del testo qualsiasi, nel nostro caso abbiamo scritto:

```
Ciao a tutti da WMLScript.it
Questo è il corso J2ME
scritto da Matteo Zinato
```

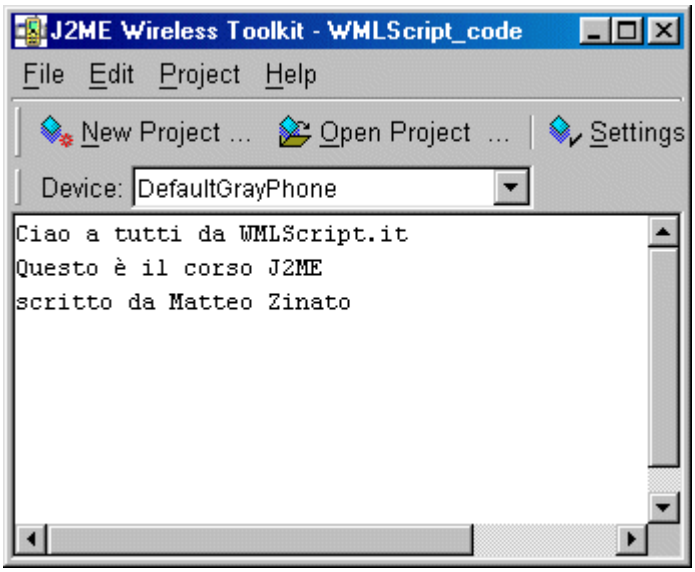
A questo punto vediamo il codice per la lettura del file:

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


public class LeggiFile extends MIDlet {

  public LeggiFile() {}

  public void startApp() {
    int c;
    InputStream is = null;

    try {
      is = getClass().getResourceAsStream("/file.test");

        if (is != null) {
          while ((c = is.read()) != -1) {
            if (c == '\n')
              System.out.println("");
            else
              System.out.print( (char)c );
          }
          is.close();
        } else {
         System.out.println("Errore nella lettura");
        }
      } catch (java.io.IOException ex) {
    }
  }

  public void pauseApp() {}

  public void destroyApp(boolean unconditional) {}

}
```

Il codice è molto semplice, non si fa altro che aprire il file come uno stream di dati in ingresso, leggerne il contenuto byte per byte e visualizzarlo sulla console, facendo attenzione ai caretteri di a

capo riga (\n).

Il risultato sulla console non è altro che la visualizzazione del contenuto del file:



Corso J2ME > Utilizzare RMS per archiviare le informazioni
scritto da Matteo Zinato

Molte delle persone che cominciano a programmare con J2ME ben presto chiedono se e come è possibile utilizzare dei database con J2ME per poter salvare e poi recuperare delle informazioni da un'applicazione. La risposta è sempre del tipo: "Si, si può, ma...".

J2ME dispone di un meccanismo che prende il nome di RMS (record management sistem), che ci permette di memorizzare informazioni in modo persistente e recuperarle in seguito, con un sistema record-oriented, simile cioè ad un database con una sola tabella e una sola chiave. La memorizzazione delle informazioni è completamente gestita del dispositivo che decide dove eseguire la memorizzazione (che dipende ovviamente dalle sue capacità) e ne gestisce l'integrità durante il normale utilizzo.

Le classi del package RMS gestiscono di fatto la comunicazione tra la memoria del dispositivo, che non è gestibile direttamente da una MIDlet, e l'applicazione che intende utilizzarla. RMS infatti utilizza le primitive del sistema operativo del cellulare o del palmare per scrivere e leggere le informazioni.

La classe principale da utilizzare per il nostro intento è RecordStore, che gestisce proprio i record di informazioni. Nell'esempio vediamo le operazioni principali:

- Apertura
- Scrittura
- Lettura
- Chiusura
- Cancellazione
- Confronto - ordimaneto

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

public class RMSRecord extends MIDlet{

  private RecordStore rs = null;
  // La variabile RecordStore, idispensabile per tutte le operazioni
  static final String RECORD_STORE = "provarms";
  // Il nome del record che utilizziamo in questo esempio

  public void destroyApp( boolean unconditional )
  { }

  public void startApp()
  {
    destroyApp(false);
    notifyDestroyed();
  }

  public void pauseApp()
  { }

  public RMSRecord(){
    openRMS();
    writeRMS();
```

```
    readRMS();
    closeRMS();

openRMS();
sortingReadRMS();
closeRMS();

    deleteRMS();
  }
```

Fermiamoci un momento sul costruttore: al suo interno vengono chiamati alcuni metodi, implementati in seguito, che non fanno altro che creare un record, scriverlo, leggerlo e chiuderlo per poi riaprirlo, leggerlo in modo ordinato e chiuderlo. Infine il record viene cancellato.

```
  // APERTURA DEL RECORD
  public void openRMS(){
    try{
      rs = RecordStore.openRecordStore(RECORD_STORE, true );
    }catch (Exception e){
      System.err.println(e.toString());
    }
  }

  // CHIUSURA DEL RECORD
  public void closeRMS(){
    try{
      rs.closeRecordStore();
    }catch (Exception e){
      System.err.println(e.toString());
    }
  }

  // CANCELLAZIONE DEFINITIVA DEL RECORD
  public void deleteRMS(){
    if (RecordStore.listRecordStores() != null){
      try{
        RecordStore.deleteRecordStore(RECORD_STORE);
      } catch (Exception e){
        System.err.println(e.toString());
      }
    }
  }

  // SCRITTURA DEL RECORD
  public void writeRMS()
  {
    String[] sData = {"Corso",
                      "J2ME",
                      "MIDlet",
                      "Di",
                      "WMLScript.it",
                      "Scritto da Matteo Zinato"};
    try{

      byte[] record;

      for (int i = 0; i < sData.length; i++){

        // ottiene dalle singole stringhe
```

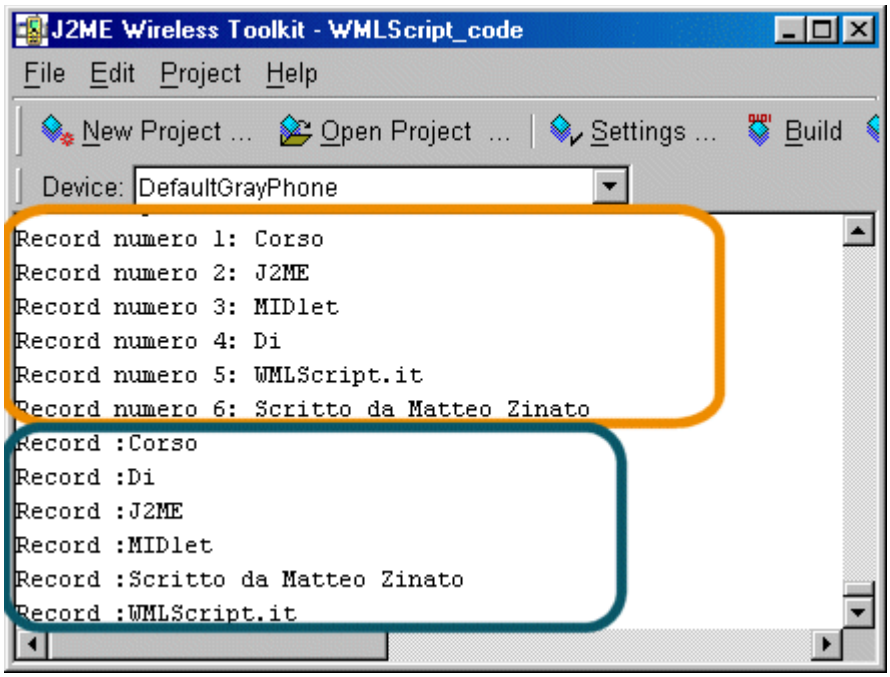
```
                // il loro equivalnete in bytes
        record = sData[i].getBytes();
        rs.addRecord(record, 0, record.length);

      }//for
    } catch (Exception e) {
      System.err.println(e.toString());
    }
  }

  // LETTURA DEL RECORD
  public void readRMS(){
    try{
      // attenzione alla dimensione del vettore di bytes
      byte[] recData = new byte[100];
      int dataLen;

      for (int i = 1; i <= rs.getNumRecords(); i++){

        rs.getRecord(i, recData, 0);
        dataLen = rs.getRecord(i, recData, 0);

        String str = new String(recData, 0, dataLen);
        System.out.println("Record numero " + i + ": " + str);

      }//for

    } catch (Exception e) {
      System.err.println(e.toString());
    }
  }

  // LETTURA DEI RECORD IN MODO ORDINATO
  public void sortingReadRMS(){
    try{
      if (rs.getNumRecords() > 0){

        SortCompare comp = new SortCompare();
        RecordEnumeration re=rs.enumerateRecords(null, comp, false);
        // re contiene ora tutti i record ordinati alfabeticamente

        while (re.hasNextElement()){
          String str = new String(re.nextRecord());
          System.out.println("Record :" + str);
        }
      }
    } catch (Exception e) {
      System.err.println(e.toString());
    }
  }

}// fine midlet
```

Nell'ultimo metodo vediamo la lettura con ordimento alfabetico, che viene sostanzialmente eseguita dalla classe **SortCompare**. SortCompare però è un'interfaccia e deve quindi essere implementata. In questo modo è chi realizza l'applicazione a stabilire i criteri di ordinamento, con la possibilità di creare più classi con più criteri.

Nel nostro caso vediamo un semplice ordinamnto alfabetico tra stringhe.

```
class SortCompare implements RecordComparator{

  // METODO PRINCIPALE PER IL CONFRONTO DI 2 ARRAY DI BYTE
  public int compare(byte[] rec1, byte[] rec2){
    String str1 = new String(rec1);
    String str2 = new String(rec2);

    int result = str1.compareTo(str2);
    if (result == 0)
      return RecordComparator.EQUIVALENT;
    else if (result < 0)
      return RecordComparator.PRECEDES;
    else
      return RecordComparator.FOLLOWS;
  }
}
```

Il risultato anche stavolta è visibile sulla console. Nell'immagine contornato di arancione la prima lettura semplice, in blu la lettura con ordinamento alfabetico.



Corso J2ME > Effettuare connessioni HTTP
scritto da Matteo Zinato

Fino ad ora abbiamo visto realizzare applicazioni che utilizzavano risorse interne all'applicazione stessa, che risiedevano quindi sul dispositivo che eseguiva il codice. Se ci pensiamo però questo può essere molto limitativo, non si sfrutta il potenziale di un cellulare che nasce di fatto per comunicare col il resto del mondo.

J2ME mette a disposizione l'oggetto **Connector** che permette di effettuari delle connessioni generiche, utilizzando qualsiaisi tipo di schema (http, ftp ecc) a qualsiasi indirizzo. E di pochi giorni fa la notizia che sun ha sviluppato delle API che permetto di inviare SMS con J2ME, semplicemente utilizzando l'oggetto Connector, collegandosi attraverso uno schema sms://+{numero_di_telefono}.

In questa lezione ci limiteremo ad una semplice connessione http, invocando una pagina web (nel nostro caso una sevlet java, giusto per restare in casa sun).
Ecco il codice della MIDlet:

```
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import java.io.IOException;
import java.io.InputStream;
public class MakeHttpConnection extends MIDlet {

  private String URL =
  "http://localhost:8080/j2me/servlet/CallServletGET"+
  "?account=matteo&password=wmlscript";

  public void startApp() {

    // crea un normale oggetto sctring buffer
    // che conterra la risposta http
    StringBuffer buffer = new StringBuffer();
    HttpConnection c = null;

    try {

      System.out.println("Connessione in corso .....\n\n");

      // apre la connessione http all'indirizzo specficato sopra
      c = (HttpConnection)Connector.open(URL);

      // una volta che la connessione è stabilita
      // apre un input stream
      InputStream is = c.openInputStream();

      int ch;

      // legge byte per byte l'inputStream e 'appende'
      // i singoli caratteri nel buffer
      while ((ch = is.read()) != -1) {
        buffer.append( (char)ch );
      }

      // chiude l'InputStream.
      is.close();
```

```
      // chiude la connessione
      c.close();

      System.out.println( "Ecco cosa ho letto dalla ");
      System.out.println( "pagina web: \n\n" + buffer.toString() );

      pauseApp();

    }
    catch ( IOException e ) {

    }

  }

  public void pauseApp() {

    System.out.println("In pauseApp .....\n\n");
    destroyApp( true );
    notifyDestroyed();

  }

  public void destroyApp( boolean unconditional ) {

        System.out.println("In destroyApp .....\n\n");

  }

}
```

Abbiamo appena visto come effettuare una semplice chiamate ad una pagina web con parametri passati nell'url, quello che viene chiamato metodo get. La classe HTTPConnection permette di personalizzare la richiesta, parteno dal metodo (post o get) al tipo di contenuto. Per chi fosse interessato ad un utilizzo massiccio di questa classe, invitiamo a consultare i javadoc di HTTPConnection.

Come detto sopra come pagina web di risposta abbiamo utilizzato una servlet java, di cui in seguito riportiamo il codice per completezza, ma è bene specificare che è possibile utilizzare qualsiasi tecnologia di scripting lato server per inviare del contunuto leggibile per la nostra MIDlet.

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class CallServletGET extends HttpServlet{

  protected void doGet(HttpServletRequest req,
                       HttpServletResponse res)
                       throws ServletException, IOException
  {
    // legge i parametri dalla MIDlet
    String acct = req.getParameter("account"),
           pwd = req.getParameter("password");

    if (acct == null || pwd == null){
      res.sendError(res.SC_BAD_REQUEST, "Parametri mancanti");
      return;
    }

    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    out.print("acct:" + acct + "\npwd:" + pwd);
    out.close();
  }


  public String getServletInfo(){
    return "Corso J2ME - WMLScript.it";
  }
}
```

Corso J2ME > Packaging delle applicazioni
scritto da Matteo Zinato

L'ultimo step logico per lo sviluppo di un'applicazione J2ME completa consistere nell'effettuare il **Packaging**, che significa creare una sorta di file eseguibile che è possibile distribuire agli utenti di dispositivi abilitati a java.
Il file eseguibile di cui si parlava in realtà è un file con estensione **JAR** (Java Archive File), che come dice il nome stesso non è altro che un archivio java di tutti i file di un'applicazioni (.class). Un file jar continene un descrittore chiamato **manifest.mf** che possiede le informazioni sul file jar, il contenuto e la vesione con cui è stato sviluppato.
Ecco per esempio il manifest di una semplice applicazione con una sola MIDlet HelloWorld

```
MIDlet-1: HelloWorld, , HelloWorld
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
Created-By: 1.4.0 (Sun Microsystems Inc.)
MicroEdition-Profile: MIDP-1.0
MIDlet-Name: WMLScript_code
MIDlet-Vendor: WMLScript.it
```

Di fatto il file jar sarebbe sufficiente per il corretto funzionamento di un'applicazione, ma ormai è prassi includere anche un file **JAD** (Java Application Descriptor) che contiene solo ed esclusivamente informazioni sulle midlet dell'applicazione. Il file jad è stato introdotto per informare un dispositivo sul contenuto del file jar prima che questo venga eseguito, in modo da filtrare eventuali applicazioni non supportate. Inoltre attraverso il file jad è possibile passare dei parametri all'applicazione senza cambiare dover metter mano ai file sorgenti dell'applicazine.
Ecco un esempio di jad, sempre riferito al codice di HelloWorld:

```
MIDlet-1: HelloWorld, , HelloWorld
MIDlet-Jar-Size: 100
MIDlet-Jar-URL: WMLScript_code.jar
MIDlet-Name: WMLScript_code
MIDlet-Vendor: WMLScript.it
MIDlet-Version: 1.0
```

Le informazioni contenute nel file jad possono essere reperite da codice con l'istruzione

```
getAppProperty("nome_property")
```

ad esempio

```
getAppProperty("MIDlet-Vendor")
```

restituisce una stringa "WMLScript.it".

Ecco dunque come si fa a distribuire le proprie midlet al pubblico, è sufficiente il consentire il download dei due file, jar e jad. L'istallazione sul dispositivo dipenderà poi dal dispositivo stesso.