

Dev.

La rivista che ti insegna a programmare

**Programmo
Subito**

Nullable types in .NET 2.0

Utilizzare le OpenGL con Qt

GPS Comunicazione e localizzazione

- Localizzazione: dispositivi e servizi
- 38.991 N 8.937 E - Instant Message in a bottle e gli altri
- Trasformare un laptop in GPS
- GPSTK, una libreria per processare dati GPS

DB

Note di Data Warehouse:
Tecniche di naming convention

JAVA

SwiXml: un tool Open Source
per realizzare GUI Java

RICERCA OPERATIVA

Ottimizzazione lineare con il
metodo del simplesso

PHP

Debug Remoto via TCP/IP in PHP

VISUAL BASIC

Sviluppare applicazioni di
riconoscimento vocale



Linux: • Sviluppo di applicazioni J2ME
in ambiente Linux

Linux: • Globus Toolkit 3

Linux: • Programmare per la
TelevisoneDigitale Terrestre
in ambiente Open Source



GRUPPO EDITORIALE
INFOMEDIA

THE SOFTWARE SIDE OF YOUR MIND

ISSN 1124-5468



9 771124 546002 5 0134

ISSN: 1124-5468

Sped. abb. post 45% articolo 2 comma 20/B legge 662/96 - Filiale di Bologna

di Roberto Casarini
rcasarini@infomedia.it

Da informatici a operai

È da tempo che incontro amici e colleghi informatici che sostengono la stessa cosa: gli informatici sono sottopagati. La domanda sorge spontanea: "quanto guadagna un informatico?". In primo luogo non si può generalizzare e andrebbe specificato meglio cosa si intende per "informatico". Ad oggi l'etichetta di informatico viene attribuita a tutti coloro che conoscono e usano con dimestichezza terminologie aliene ai non addetti: basta parlare per dieci minuti in "informaticese" ed il gioco è fatto. Software engineer, web designer, web developer, consulente informatico, database administrator, network administrator, system engineer: tante categorie ricercate nel mercato del lavoro. Nonostante il bagaglio tecnico, faticosamente acquisito negli anni per svolgere una professione in auge, nonché affascinante, non sempre retribuzioni e gratificazioni sono adeguate. Si tratta solo di un problema di mancata meritocrazia o c'è dell'altro?

Sarebbe auspicabile avere un confronto con voi lettori su un malessere che si sta manifestando sempre più diffusamente, in un settore potenzialmente ancora fiorente, che tuttavia è spesso capace solo di "apprezzare" gli informatici come "manovalanza".

Capita sovente di parlare con colleghi che sono "costretti" ad eseguire mansioni non attinenti alla propria competenza specifica, e ciò esclusivamente per necessità.

Per non parlare di chi oggi si accinge ad entrare nel mondo del lavoro come informatico, con rapporti di lavoro che quasi sempre iniziano come stage, un percorso divenuto quasi scontato. Piccole (e non) aziende informatiche che si appoggiano a un continuo riciclo di tirocinanti. E i neolaureati sono consapevoli, loro malgrado, che una collaborazione stabile non è scontata, perché così sono le regole del "gioco". Le aziende sostengono che in questo modo vengono offerte possibilità di inserimento e soprattutto vengono garantite preziose esperienze di lavoro. Allora perché non far pagare il tirocinante per questo genere di "formazione" equiparandola a una sorta di master? Qualche azienda non si sarà di certo fatta sfuggire l'idea... Qual è la responsabilità da parte del mondo della formazione, sia secondaria sia delle Università, il cui ruolo di interfaccia col mondo del lavoro è sempre carente e non promuove rapporti di lavoro non precari?

E purtroppo la crisi economica, sta minando sempre più le motivazioni di intraprendere la professione di programmatore, anche per il ricorso a forme di outsourcing che permettono l'accesso a competenze di ottimo livello ma con retribuzioni... al ribasso. Un fattore determinante per comprendere il clima dell'attuale panorama economico.

Queste riflessioni, ahimè note a molti informatici, spesso suscitano una sola risposta, comune e perentoria: "Basta, cambio mestiere!"

Basterà la passione, mai deleteria, per continuare a ritenere che, nonostante tutto, "l'ombrello informatico" rimane pur sempre una bellissima professione?

Roberto Casarini



Sei agli inizi? Non riesci a risolvere i primi problemi? Niente paura! Iscriviti alla mailing-list Infomedia!

programma: sabbat@infomedia.it
e linux@infomedia.it

I listati e i programmi di utilità che corredano gli articoli sono prelevabili gratuitamente dal sito <http://www.infomedia.it/DEV/> oppure da <http://ftp.infomedia.it/pub/DEV/Listati/>



Lo speciale di questo numero: GPS, comunicazione e localizzazione

- 11 Localizzazione: dispositivi e servizi di Marco Fabbri
- 15 38.991 N 8.937 E - Instant Message in a bottle e gli altri di Davide Carboni e Stefano Sarma
- 20 Trasformare un laptop in GPS di Ambrogio Galbusera
- 25 GPSTK, una libreria per processare dati GPS di Roberto Casarini

Programmazione

- 28 Cellulari - Il protocollo AT+ dei telefoni cellulari. Comandi speciali Siemens: ancora sull'Organizer (parte 20) di Vito Vessia
- 33 Database - Note di Data Warehouse: Tecniche di naming convention (parte 1) di Massimo Cenci
- 37 Java - SwiXml: un tool Open Source per realizzare GUI Java (parte 1) di Nunzio Fiore
- 40 Java - The table show with 'DisplayTag' di Francesco Tinti
- 43 Ricerca Operativa - Ottimizzazione lineare con il metodo del semplice di Vittorio Rigamonti
- 51 Sicurezza - Buffer overflow e dintorni: programmazione sicura (ULTIMA PARTE) di Gianluca Insolubile
- 55 PHP - Debug Remoto via TCP/IP in PHP di Max Arcidiacono
- 58 Visual Basic - Sviluppare applicazioni di riconoscimento vocale di Lorenzo Vandoni
- 60 Visual Objects - PDF995 e Report Pro di Danilo Giuliani
- 62 Web - Digital Divide: what's new? what's not? (and so what?) (parte 1) di Cesare Zavalloni

Rubriche

- 8 Extended Mode - S.C.A.R. Squadra Corse Alfa Romeo a cura di Stefano Arcidiacono
- 10 CrackMe web edition - Accesso riservato a... a cura di Fabio Scagliola
- 65 Normative - La tutela delle pagine web (parte 4) di Massimo Farina
- 90 Game Dev - Costruire un'applicazione in J2ME utilizzando MIDP2 di Mauro Pesci
- 98 Horizon - L'insostenibile leggerezza dell'essere - di Luigi Morelli

Programmo Subito

Numero 87

- 68 **Livello 3** Nullable types in .NET 2.0 di Gian Maria Ricci
- 71 **Livello 3** Utilizzare le OpenGL con Qt di Fabrizio Angius
- 76 **Livello 1** L'evoluzione del Web e la nascita delle Web Application di Umberto Annunziata

Linux Subito

Numero 66

- 79 **Livello 1** Sviluppo di applicazioni J2ME in ambiente Linux di Stefano Sanna
- 83 **Livello 2** Globus Toolkit 3 - di Angelo Raffaele Bruno
- 86 **Livello 1** Programmare per la Televisione Digitale Terrestre in ambiente OpenSource - di Fabio Ramacciani

Ai lettori

Linux 001 a cura di Remigio Darino
Linux e i suoi amici

32

E-ducational a cura di Luigi Morelli
Algoritmi di ordinamento

36

Doktor Morb a cura di Luigi Morelli
Matematica calcistica

54

Java ME World a cura di Stefano Sanna
Bigger is better!

70

Free Reading a cura di Lorenzo Vandoni
Building Accessible Websites

82

Dev

Numero 134
Novembre 2005
Anno XIII

Direttore Responsabile
Martaletizia Mori (mori@infomedia.it)

Coordinamento Editoriale
Roberto Casarini (casarini@infomedia.it)

Collaboratori
Fabrizio Angius, Umberto Annunziata
Stefano Arcidiacono, Max Arcidiacono
Angelo Raffaele Bruno
Massimo Cenci, Remigio Darino
Marco Fabbri, Massimo Farina
Nunzio Fiore, Ambrogio Galbusera
Gianluca Insolibile
Luigi Morelli, Mauro Pesci
Fabio Ramacciani, Gian Maria Ricci
Vittorio Rigamonti, Stefano Sanna
Fabio Scagliola, Francesco Tinti
Lorenzo Vandoni, Vito Vessia
Cesare Zavalloni

Grafica e Impaginazione
Manola Greco (greco@infomedia.it)

**GRUPPO EDITORIALE
INFOMEDIA**

Gruppo Editoriale Infomedia srl
Via Valdera P.116 56038 Ponnacco (PI)
Tel. 0587 73 64 60 (r.a.)
Fax 0587 73 22 32
www.infomedia.it - red_dev@infomedia.it

Direzione
Natale Fino (fino@infomedia.it)

Divisione Libri
Lisa Vanni (vanni@infomedia.it)

Marketing & Advertising
Tel. 0587 73 64 60
marketing@infomedia.it

Supporto Sistemistico
webmaster@infomedia.it

Amministrazione
Sara Mattioli (mattioli@infomedia.it)

Segreteria
Enrica Nanni (nanni@infomedia.it)

Stampa
Betagraf Spa - Puno di Argelato (BO)

Distribuzione
Parrini & C. (Roma)

Ufficio Abbonamenti
Tel. 0587 73 64 60
Fax 0587 73 22 32
abbonamenti@infomedia.it
http://online.infomedia.it

Si prega di inviare i comunicati stampa e gli inviti stampa per la redazione all'indirizzo: comunicatistampa@infomedia.it

La riproduzione con qualsiasi mezzo di illustrazioni e di articoli pubblicati sulla rivista, nonché la loro traduzione, è riservata e non può avvenire senza esplicita autorizzazione dell'editore. I manoscritti e le illustrazioni inviati alla redazione non saranno restituiti anche se non pubblicati e l'Editore non si assume alcuna responsabilità per il caso che si tratti di esemplari unici.

DEV: Developing Software & Software
è una rivista del Gruppo Editoriale Infomedia S.p.A.

Direzione e Amministrazione
Via Valdera P. 116 Ponnacco
Registrazione presso il Tribunale di Pisa
n. 25 del 25-31-1999

www.infomedia.it - The Software Side Of Your Mind

Costruire un'applicazione in J2ME utilizzando MIDP2

Impariamo a scrivere un modello di applicazione stabile per le nostre applicazioni J2ME sfruttando appieno gli strumenti offerti dal profilo MIDP2

Il linguaggio Java, ormai supportato dalla maggior parte dei dispositivi portatili, ha donato nuove possibilità alla fantasia e all'ingegno dei programmatori tanto che, in questi ultimi anni, stiamo assistendo ad una graduale ed inesorabile crescita del business dei videogiochi, in particolar modo, per cellulari.

Le novità tecniche introdotte con la versione delle MIDP 2.0 hanno notevolmente favorito questo sviluppo mettendo a disposizione del programmatore un insieme molto completo di strumenti, rispetto alla versione 1.0. In precedenza, ad esempio, non era possibile far suonare i file musicali nei propri programmi se non si utilizzavano gli SDK proprietari forniti dalla casa produttrice del dispositivo o costosi tool commerciali. In più, argomento da tenere in grande considerazione, non dovremo preoccuparci di reperire gli strumenti di lavoro adeguati tra quelli commerciali data la possibilità di ottenerne uno, molto valido, in forma gratuita. Si tratta di NetBeans un ambiente di sviluppo molto completo e potente di cui parleremo nei prossimi articoli. Anche nel caso di J2ME gli argomenti da trattare non sono pochi, per cui abbiamo deciso di iniziare una serie approfondita che ci guiderà fino alla realizzazione di un videogioco completo per cellulari. L'obiettivo principale di questo articolo, come dice

il titolo, è quello di arrivare a definire una forma standard per il codice che terrà in piedi tutta la struttura delle nostre applicazioni. Vedremo anche come temporizzare i fotogrammi in modo che vengano visualizzati con una cadenza costante e regolare su tutti i dispositivi.

Introduzione

J2ME è un framework che si rivolge direttamente ad apparecchi con capacità molto limitate. Molti di essi come cellulari e cercapersone non permettevano, in passato, di installare altri software in aggiunta a quelli precaricati durante il processo di produzione. Con la sua introduzione, J2ME, ha fatto uscire questi dispositivi dalla loro originale staticità.

Ovviamente, per funzionare con basse capacità di calcolo e memorizzazione, è stato necessario sottostare ad alcuni compromessi. J2ME, infatti, non è altro che un sottoinsieme della API J2SE, quindi, molte delle sue caratteristiche non sono più disponibili come, ad esempio, AWT (Abstract Window Toolkit) inutilizzabile su display di dimensioni troppo ridotte. L'idea principale di Sun è stata quella di creare una piattaforma comune a tutti i dispositivi con un set di API orientato a fornire una certa gamma di funzionalità basilari. Tramite il profilo MIDP1, infatti, era possibile solamente la realizzazione di applicazioni, molto semplici.

Con il progredire delle tecnologie, questi terminali si sono evoluti possedendo caratteristiche sempre più sofisticate. Quale poteva essere la direzione intrapresa da tali progressi se non quella del crescente potenziamento delle funzionalità multimediali? Per questo nasce MIDP2 orientato a fornire

il supporto per le componenti essenziali dei videogiochi, ossia, sprite, suoni e animazioni oltre che alla gestione migliorata della rete. Vediamo quindi come utilizzarlo al meglio.

Di cosa abbiamo bisogno

Come già detto, in questa puntata, si vuole arrivare a scrivere le classi Java da utilizzare come template di base per impostare qualunque videogioco. Raggiungere questo obiettivo significa stabilire una struttura molto generica e schematica delle applicazioni che realizzeremo. È necessario, in primo luogo, tenere in considerazione il

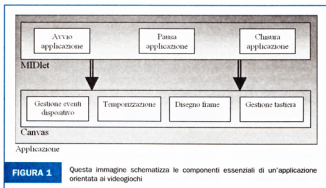


FIGURA 1

Questa immagine schematizza le componenti essenziali di un'applicazione orientata ai videogiochi

LISTATO 1 Il codice relativo alla classe MIDlet

```

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

public class HelloWorld extends MIDlet
{
    HelloWorldCanvas helloWorldCanvas;

    private boolean init = false;
    private boolean pausa = false;

    public HelloWorld(){
    }

    public void startApp()
    {
        if (init == false) init();

        // Recupero il puntatore al display correntemente utilizzato
        try
        {
            if (pausa == false)
            {
                helloWorldCanvas.start();
            }
            else
            {
                // Rilancio conseguentemente ad una pausa
                // l'esecuzione del Canvas
                helloWorldCanvas.restart();
            }
            pausa = false;
        }

        // Imposto il puntatore al display
        Display.getDisplay(this).setCurrent(helloWorldCanvas);
    }
    catch(Exception e)
    {
        // Impossibile avviare l'applicazione
    }

    public void pauseApp()
    {
        // Fermo l'esecuzione del Canvas
        helloWorldCanvas.pause();
        pausa = true;
    }

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

    // Inizializzazione
    private synchronized void init()
    {
        if (!init)
        {
            try
            {
                helloWorldCanvas = new HelloWorldCanvas(this);
                init = true;
            }
            catch (Exception e)
            {
                init = false;
            }
        }
    }
}

```

metodo utilizzato per interfacciare l'applicazione con gli eventi generati dal dispositivo ospite. Data l'ampia gamma di dispositivi per i quali J2ME è stato progettato, l'insieme degli eventi che possono essere intercettati dall'applicazione sono relativamente pochi. Come vedremo meglio poi, dovremo focalizzare il nostro interesse su tre messaggi che possono essere lanciati in altrettante situazioni durante il ciclo di vita dei programmi J2ME. Il primo caso si verifica, ovviamente, all'avvio dell'applicazione e ad ogni ripristino dell'esecuzione dopo una messa in pausa. Il secondo avviene quando il dispositivo avverte che l'applicazione sarà messa in pausa, per via, magari, di una telefonata entrante. L'ultimo viene inviato al momento della richiesta di chiusura del programma. E' la classe *MIDlet* ad implementare i metodi necessari ad assolvere a questi compiti e verrà studiata tra poco. Ora, invece, è importante decidere quali saranno le caratteristiche che dovrà possedere la seconda classe, o *Canvas*, che gestirà il gioco vero e proprio. Dovranno essere sicuramente aggiunti alcuni metodi per controllare l'esecuzione del programma secondo i messaggi intercettati dalla classe *MIDlet*. Dopodiché sarà la volta del meccanismo di temporizzazione necessario per regolarizzare i tempi di esecuzione del programma. Infine, non mancheranno un gestore degli eventi della tastiera ed un metodo

che richiamato ad ogni passo dell'esecuzione disegnerà sul display i fotogrammi dell'azione. In **Figura 1** ho rappresentato una schematizzazione dei moduli che intendiamo realizzare e dei rispettivi componenti interni.

Preparare il sistema per lo sviluppo

Siamo arrivati ad un passo dalla fase di scrittura del codice. Ovviamente si rivela necessario disporre degli adeguati strumenti per portare a termine lo sviluppo. Innanzitutto dobbiamo aver installato sul nostro PC l'ultima versione del pacchetto J2SE SDK reperibile direttamente sul sito della Sun www.sun.com. Da questo momento il PC è in grado di compilare ed eseguire quasi tutti i tipi di programmi Java. Dico quasi poiché il secondo passo da effettuare è quello di scaricare dallo stesso sito il pacchetto *J2ME Wireless Toolkit*. Verranno, quindi, installati tutti i componenti necessari allo sviluppo specifico per cellulari compresi gli emulatori per eseguire i programmi direttamente sul PC. Tra le utility è presente un programmino, chiamato *KToolbar*, che facilita le operazioni più comuni che si effettuano con i progetti. Come mostrato in **Figura 2** il tool prevede, infatti, alcuni per creare i

progetti, compilare i sorgenti, eseguirli sull'emulatore e, infine, generare i pacchetti da distribuire. KToolbar mantiene i progetti nella propria directory di installazione; se avete installato la versione di 2.2 di Wireless Toolkit la locazione di default è C:\WTK22, mentre la sottodirectory per i progetti è chiamata *apps*. Ogni progetto è organizzato in una serie di sottocartelle contenenti i vari tipi di file che prendono parte allo sviluppo del programma. Per i nostri scopi ci basta lavorare all'interno della *src*, dove sono raggruppati tutti i sorgenti, e memorizzare nella *res* tutti i vari file di risorsa come immagini, suoni ecc. Una volta terminato il lavoro e compilato senza errori, nella directory *bin*, troveremo i file JAR e JAD. Per scrivere i sorgenti il Wireless Toolkit non dispone di alcun editor per cui dovremo utilizzare un programma qualsiasi, quello che più ci aggrada. Ora che sappiamo tutto possiamo procedere con la creazione di un nuovo progetto chiamato *HelloWorld*.

La classe principale: MIDlet

Per scrivere la *MIDlet* abbiamo bisogno di importare, nel nostro progetto, i seguenti package:

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

Il primo contiene semplicemente la definizione della classe astratta *MIDlet* la quale dovrà essere ereditata dalla nostra classe *HelloWorld*. Il secondo, invece, ha le definizioni di tutte le funzionalità base per la gestione dell'interfaccia grafica. Dopo questa operazione possiamo procedere con la dichiarazione della classe che intendiamo scrivere:

```
public class HelloWorld extends MIDlet
```

La classe *MIDlet* definisce anche alcuni metodi astratti

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

che rappresentano il punto di ingresso all'applicazione da parte del sistema. Essi vengono richiamati, come detto sopra, in relazione alle diverse situazioni che si possono verificare durante il ciclo di vita della *MIDlet* quali l'avvio del programma, la messa in pausa e la sua terminazione. Senza la necessità di utilizzare gli oggetti propri dell'interfaccia grafica, la *MIDlet* si riduce al ruolo di "tramite" tra il sistema e la nostra classe *Canvas*. Analizzando il codice dei metodi nel Listato 1 possiamo capire come può essere organizzato, in maniera molto semplice, un tale sistema. Dato che il metodo *startApp()* viene lanciato alla prima esecuzione del programma e ad ogni suo ripristino da una condizione di pausa, si rivela necessario inserire un controllo per determinare a quale delle due situazioni corrisponde tale chiamata. Questo controllo è effettuato tramite il test su una variabile privata che ha funzione di segnalatore per indicare se la classe *Canvas* è stata precedentemente messa in pausa o meno. A seconda dello stato della variabile *pausa* il programma decide se risvegliare il *Canvas* oppure lanciarne l'inizializzazione. Il metodo *pauseApp()* contiene solamente la chiamata



FIGURA 2 Una vista della KToolbar e i suoi menu utili allo sviluppo

all'apposita funzione della classe *Canvas* per sospendere l'esecuzione ed un'assegnamento alla variabile che segnala l'avvenuta messa in pausa. Il metodo *destroyApp()* notifica che l'esecuzione dell'applicazione è conclusa e che il *garbage collector* può procedere a liberare la memoria da tutti gli oggetti allocati. L'ultimo metodo, *init()*, serve a creare un'istanza della classe *Canvas* ed a notificare l'avvenuta inizializzazione del programma.

Il nucleo dell'applicazione: la classe Canvas

Se la classe *MIDlet* rappresenta l'interfaccia dell'applicazione nei confronti del dispositivo, la classe *Canvas* è la sede operativa di questa. Essa eredita dalla *GameCanvas* le funzioni basilari come il disegno off-screen e la rilevazione degli eventi scaturiti dalla tastiera e dal puntatore, se supportato dal dispositivo. Pertanto, nel file *HelloWorldCanvas.java*, scriveremo tutto il codice di gestione del gioco; il Listato 2 ne presenta il sorgente completo che provvediamo subito a commentare. Come in tutte le applicazioni Java, le prime linee di codice da scrivere riguardano l'inclusione dei package. In questo caso abbiamo:

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

Dato che non utilizzeremo gli oggetti grafici standard come form pulsanti e così via la prima inclusione è necessaria, sostanzialmente, per accedere alle funzioni di gestione delle immagini. La seconda, invece, include tutte ciò che è necessario alla realizzazione dei giochi come gli *sprite*, i *layer* e gli sfondi in *tilemapping*. La prossima riga

```
public class HelloWorldCanvas extends GameCanvas implements Runnable
```

definisce la classe *Canvas* chiamata *HelloWorldCanvas*. Il costruttore richiede, come parametro, un riferimento alla classe *MIDlet* per eventuali chiamate ai suoi metodi

```
HelloWorldCanvas (HelloWorld HelloWorldMIDlet)
```

LISTATO 2 Il codice della classe Canvas (continua)

```

import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
import javax.microedition.media.*;
import java.io.*;
import java.util.*;

public class HelloWorldCanvas extends GameCanvas implements Runnable
{
    Graphics g;
    HelloWorld istanza_midlet;

    private final int MILLIS_PER_FRAME = 40;
    private int LARGHEZZA_DISPLAY;
    private int ALTEZZA_DISPLAY;
    private boolean DISPLAY_COLORE;
    private int PROFONDITA_COLORE;
    private int PROFONDITA_ALPHA;

    private volatile Thread aniThread = null;

    // Costruttore della classe Canvas
    HelloWorldCanvas (HelloWorld HelloWorldMIDlet)
    {
        super(true);

        // Memorizzazione del riferimento alla
        // classe MIDlet
        istanza_midlet = InterfacciaMIDlet;

        // Impostazione della modalità a schermo pieno
        setFullScreenMode(true);

        // Recupero delle dimensioni del display
        LARGHEZZA_DISPLAY = getWidth();
        ALTEZZA_DISPLAY = getHeight();

        Display d = Display.getDisplay(istanza_midlet);

        // Determinazione del tipo di display
        // Colore/Toni di grigio
        DISPLAY_COLORE = d.isColor();

        // Determinazione del numero dei colori
        // visualizzabili del display
        PROFONDITA_COLORE = d.numColors();

        // Determinazione del numero dei livelli
        // di trasparenza utilizzabili
        PROFONDITA_ALPHA = d.numAlphaLevels();

        // Recupero del riferimento all'oggetto
        // grafico
        g = getGraphics();
    }

    // Metodo richiamato dalla classe
    // MIDlet per lanciare l'applicazione
    public void start()
    {
        // Creazione e avvio di un nuovo
        // thread
        aniThread = new Thread(this);
        aniThread.start();

        // Lancio dell'applicazione
        run();
    }

    // Richiamato dalla classe MIDlet per
    // ripristinare l'esecuzione dopo
    // un'interruzione
    public void restart()
    {
        aniThread.notify();
    }

    // Richiamato dalla classe MIDlet per
    // sospendere l'esecuzione del
    // programma
    public void pause()
    {
        try
        {
            aniThread.wait();
        }
        catch (Exception e)
        {
        }
    }

    // Richiamato dalla classe MIDlet per
    // terminare l'esecuzione del programma
    public void stop()
    {
        aniThread = null;
    }

    public void run()
    {
        Thread cThread = Thread.currentThread();

        try
        {
            while (cThread == aniThread)
            {
                long startTime = System.currentTimeMillis();

                if (isShown())1
                {
                    keyboard();
                    frame();
                    draw();
                    flushGraphics();
                }

                long endTime = System.currentTimeMillis() - startTime;

                if (endTime < MILLIS_PER_FRAME)

```

LISTATO 2 Il codice della classe Canvas (segue)

```

{
    synchronized(this)
    {
        wait(MILLIS_PER_FRAME - endTime);
    }
}
else
{
    cThread.yield();
}
}
}
catch (InterruptedException ex)
{
    // Nulla
}
}

void frame()
{
}

void keyboard()
{
    int statoTastiera = getKeyStates();

    if ((statoTastiera & LEFT_PRESSED) != 0)
    {
    }
    else if ((statoTastiera & RIGHT_PRESSED) != 0)
    {
    }
    else if ((statoTastiera & FIRE_PRESSED) != 0)
    {
    }
    else if ((statoTastiera & UP_PRESSED) != 0)
    {
    }
    else if ((statoTastiera & DOWN_PRESSED) != 0)
    {
    }
    else if (statoTastiera == 0)
    {
    }
    else
    {
    }
}

private void draw()
{
}
}
    
```

All'interno del costruttore osserviamo, poi, alcune istruzioni. La chiamata `setFullScreenMode` attiva o disattiva la modalità full screen una caratteristica, questa, molto interessante se vogliamo avere a disposizione l'intera area del display nascondendo, quindi, tutti i messaggi e/o immagini informative normalmente presenti. Da notare che attivando questa modalità non verranno visualizzati nemmeno gli eventuali elementi aggiunti nella classe `MIDlet` come, ad esempio, i menu o le caption dei tasti funzione. Se è nostra intenzione che un programma venga lanciato su diversi tipi di dispositivo è d'obbligo essere in grado di determinare le caratteristiche di ogni display. Esistono, infatti, diverse classi di display che si differenziano per le loro dimensioni in pixel e la profondità di colore che sono in grado di visualizzare. A questo scopo utilizziamo le funzioni `getWidth()` e `getHeight()` che ne restituiscono la larghezza ed altezza, mentre i metodi `isColor()`, `numColors()`, `numAlphaLevels()` della classe `Display` servono a determinare se, questo, è a colori o toni di grigio, quanti livelli di colore riesce a visualizzare e quanti livelli di trasparenza possono essere gestiti. Analizziamo ora come interfacciarci con la classe `MIDlet` rispondendo opportunamente agli eventi in precedenza menzionati. In primo luogo forniremo un'adeguata implementazione del metodo `start()`

```

anThread = new Thread(this);
anThread.start();

run();
    
```

la quale non fa altro che creare una nuova istanza di un oggetto `Thread` ed avviarla. La chiamata al metodo privato `run()` determina l'avvio del ciclo principale del gioco. Quando il dispositivo richiederà che il programma venga messo in pausa effettueremo semplicemente questa operazione, nel metodo `pause()`

```

try
{
    anThread.wait();
}
catch (Exception e)
{
}
    
```

In questo modo il `Thread`, ossia il ciclo principale, viene sospeso fino alla successiva chiamata della funzione

```
anThread.notify();
```

contenuta nel metodo `restart()`.

Al momento di concludere tutte le operazioni, il metodo `stop()`

```
anThread = null;
```

sgancia l'istanza del *Thread* in modo da poter essere deallocata dal *garbage collector*. Ciò che rimane del codice rappresenta il succo del nostro lavoro, ossia, i quattro membri della classe che vengono eseguiti ad ogni iterazione del ciclo principale. In primis osserviamo il metodo *run()*, richiamato la prima volta dal metodo *start()*, che risulta così strutturato

```
Thread cThread = Thread.currentThread();

try
{
    while (cThread == ariThread)
    {
        long startTime = System.currentTimeMillis();

        if (isShown())
        {
            keyboard();
            frame();
            draw();
            flushGraphics();
        }

        long endTime = System.currentTimeMillis() - startTime;

        if (endTime < MILLIS_PER_FRAME)
        {
            synchronized(this)
            {
                wait(MILLIS_PER_FRAME - endTime);
            }
        }
        else
        {
            cThread.yield();
        }
    }
} catch (InterruptedException ex)
{
}
```

Dopo aver controllato che il *Thread* correntemente in esecuzione è lo stesso da noi istanziato in precedenza si passa alla rilevazione del valore del timer memorizzandolo nella variabile *startTime*. Dopodiché se il nostro *Canvas* risulta visibile sul display, *isShown()*, eseguiamo le operazioni periodiche di

- Controllo dello stato della tastiera, se risultano, cioè, premuti tasti e quali, tramite il metodo *keyboard()*
- Esecuzione delle normali attività di gestione del gioco, con il metodo *frame()*
- Disegno dell'output sul buffer off-screen, con la chiamata a *draw()*
- Visualizzazione finale del fotogramma sul display tramite *flushGraphics()*

A questo punto si rilegge lo stato del timer e sottraendo a questo



FIGURA 3

Un'immagine del programma di esempio presente sul sito FTP in esecuzione sull'emulatore di default

il valore precedentemente ottenuto ci possiamo rendere conto se le operazioni effettuate hanno richiesto una quantità di tempo inferiore a quella definita nella variabile *MILLIS_PER_FRAME*. In questo caso dobbiamo fare in modo che l'ultimo fotogramma prodotto rimanga visibile per un periodo di tempo che è pari alla differenza tra il valore da noi indicato e il tempo impiegato. Per ottenere questo risultato basta fermare il programma con la chiamata alla funzione *wait()*, in questo modo

```
wait(MILLIS_PER_FRAME - endTime);
```

L'ultima operazione da effettuare è quella di permettere agli altri *Thread*, in esecuzione sul dispositivo, di operare chiamando

```
cThread.yield();
```

Conclusioni

Questo è tutto. Non si tratta di una grande quantità di codice e una volta impostato come template possiamo creare nuovi programmi completamente funzionanti semplicemente modificando i nomi delle classi ed implementando opportunamente i metodi *frame()*, *keyboard()* e *display()*.

In **Figura 3** viene mostrato l'output sull'emulatore del semplice programmino che potrete trovare sul sito FTP. Spero di aver definito un buon punto di partenza per iniziare ad esplorare esaurientemente questo nuovo mondo tutt'ora in piena evoluzione. Ricordate, però, che per noi continuano ad essere molto importanti i vostri riscontri, quindi scrivete e proponeteci i vostri consigli.

Bibliografia

- [1] Carol Hamer, "J2ME Games with MIDP2", Apress, 2004, ISBN 1-59059-382-0, www.apress.com
- [2] John W. Muchow, "J2ME Guida pratica alla programmazione di dispositivi wireless", Mc Graw Hill, 2002, ISBN 88-386-4278-8

Download del progetto

Scarica il progetto completo di questo articolo all'indirizzo: <http://ftp.infomedia.it/pub/DEV/Listati/Dev134/gamesdev.zip>

Mauro Pesci

Si occupa di Analisi e Sviluppo del software in vari ambienti e piattaforme. Ha lavorato per diverse aziende nel settore, mentre ora è impegnato in un progetto presso la HP di Roma. Attualmente studia Informatica presso l'università di Tor Vergata (Roma).