API MIDP PER L'INTERFACCIA UTENTE
I COMMAND

By Fabrizio Russo
02/12/2002

www.frusso.it/j2me

L'oggetto più alto nella gerarchia di classi messa a disposizione dalla libreria MIDP per l'interfaccia utente è la classe `javax.microedition.lcdui.Screen`**. Questo significa che l'interazione che le applicazioni basate su questo tipo di architettura avranno con l'utente sono basate su una successioni di *schermi*.**

Per ogni schermo, l'utente avrà a disposizione una serie di comandi attraverso i quali sarà possibile determinare quale sarà lo schermo successivo da presentare, quale computazione effettuare, quale richiesta effettuare in rete e così via.

Vediamo quali sono gli oggetti che la libreria MIDP mette a disposizione per poter costruire interfacce utente per le applicazioni.

Command

Gli oggetti di tipo `javax.microedition.lcdui.Command` **incapsulano il nome e l'informazione relativa alla semantica dell'operazione che essi rappresentano. Il suo uso primario è quello di presentare una scelta all'utente che, in prima battuta, può essere immaginato come una azione da intraprendere in seguito alla pressione di un tasto. Il comportamento che il comando attiva non è incapsulato nel comando stesso, ciò significa che il comando contiene informazioni circa il "comando" da intraprendere e non come eseguirlo. Il "come" eseguire un comando è definito in un oggetto che implementa l'interfaccia** `javax.microedition.lcdui.CommandListener` **associato con lo schermo che contiene il comando.**

Ogni comando contiene tre tipi di informazione: una label, un tipo ed una priorità. La label viene utilizzata per la rappresentazione grafica del comando (in pratica cosa verrà visualizzato sullo schermo), il tipo e la priorità sono utilizzati dal sistema per determinare come il comando sarà mappato con il terminale. (un elenco di possibili tipi di comandi verrà dato in seguito). Un oggetto di tipo Command viene creato utilizzando il suo costruttore

```
public Command(String label, int commandType, int priority)
```

Esistono diversi tipi di comandi predefiniti, ognuno dei quali cerca di mapparsi con uno dei tasti presenti sul teminale. Per esempio esiste il tipo di comando "EXIT" che cercherà di essere mappato con il tasto exit presente sul telefonino, qualora questo esista. Nel caso che non esista un tasto con quel significato, allora il sistema cercherà di assegnare un tasto il più "aderente" possibile alle specifiche del comando.

I comandi possono essere implemenati in ogni interfaccia che ha la semantica per attivare una singola azione (un pulsante, un item di menù, o qualche altro componente di interfaccia utente).

Mappare il comando con i tasti del terminale è compito dell'implementazione MIDP del terminale su cui gira l'applicazione e dipende anche dal numero totale di comandi presenti su uno schemo. Per esmepio, se una applicazione chiede di utilizzare più comandi di quanti tasti siano presenti su un terminale, il terminale potrebbe utilizzare un altro tipo di interfaccia (tipo un menù) per mostrare tutti i possibili comandi.

Ad esempio, supponiamo di avere di aggiungere la seguente sequenza di comandi ad una lista di marche d'auto.

```
new Command("Buy", Command.SCREEN, 1);
new Command("Info", Command.SCREEN, 1);
new Command("Back", Command.BACK, 1);
```

Un'implementazione che abbia a disposizione due soli pulsanti (come la maggior parte dei telefonini) potrebbe mappare il comando BACK al pulsante di sinistra e creare (in automatico) una lista di comandi (Menu) ed associarla al tasto di destra

Quando l'utente preme il tasto di destra (Menu) il terminale presenterà la lista di comandi associati allo schermo:

Tipologie di comandi

I tipi di comandi messi a disposizione dalla MIDP sono i seguenti:

BACK	**Consente all'utente di tornare allo schermo precedente7**
CANCEL	**Rappresenta il tasto di annullamento per un dialogo implementato dallo schermo**
EXIT	**Comando usato per uscire dall'applicazione**
HELP	**Richiesta specifica di maggiori informazioni**
ITEM	**Identifica un comando assegnabile un item facente parte dello schermo**
OK	**Risposta affermativa ad un dialogo**
SCREEN	**Comando definito dall'utente**
STOP	**Comando che annulla il processo attualmente in corso**

Attenzione. Il fatto di dichiarare un comando di un tipo piuttosto che di un altro non fa sì che l'implementazione del comando faccia quello che ci si aspetti. Per esempio, dichiarare un comando di tipo EXIT non significa che se viene premuto l'applicazione terminerà. L'implementazione del comando è demandata ad oggetti che implementino l'interfaccia `CommandListener`. La scelta del tipo di comando influirà solo sulla mappatura che il terminale effettuerà per il tipo di comando (Per esempio se sul telefono/palmare esiste un tasto predefinito per l'uscita allora molto probabilmente il terminale mapperà quel comando con quel tasto).

Procediamo ora con un esempio che illustri come creare una MIDlet che aggiunga un comando e come assegnare un comportamento ad un comando.

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


public class EsempioMidlet extends MIDlet implements CommandListener{

   Command cmdExit = null;
   Command cmdCommand1 = null;
   Command cmdCommand2 = null;
   Command cmdCommand3 = null;
   StringItem strLabelCommand = null;



   public EsempioMidlet() {
      super();
      cmdExit = new Command("Esci", Command.EXIT, 1);
      cmdCommand1 = new Command("Command1", Command.SCREEN, 1);
      cmdCommand2 = new Command("Command2", Command.SCREEN, 1);
      cmdCommand3 = new Command("Command3", Command.SCREEN, 1);
   }


   protected void startApp() {
      Form frmMain = new Form("Esempio");
      frmMain.addCommand(cmdExit);
      frmMain.addCommand(cmdCommand1);
      frmMain.addCommand(cmdCommand2);
      frmMain.addCommand(cmdCommand3);
      frmMain.setCommandListener(this);
      strLabelCommand = new StringItem("Label Command", "");
      frmMain.append(strLabelCommand);
      Display.getDisplay(this).setCurrent(frmMain);
    }


   protected void pauseApp() {}


   protected void destroyApp(boolean arg0) {}


   public void commandAction(Command cmd, Displayable disp) {
      if (cmd == cmdExit) {
        destroyApp(false); notifyDestroyed();
       } else {
        String label = cmd.getLabel();
        strLabelCommand.setText(label);
       }
   }

} // End of class
```

Questo esempio mostra un'applicazione che ha come unico obiettivo quello di mostrare la label del comando che si è selezionato.

La classe estende MIDlet (altrimenti non sarebbe una MIDlet) ed implementa l'interfaccia CommandListener in quanto vuole essere notificata dell'eventuale scelta da parte dell'utente di uno dei quattro comandi che l'applicazione crea.

Le proprietà definite a livello di classe sono le seguenti:

```
Command cmdExit = null;
Command cmdCommand1 = null;
Command cmdCommand2 = null;
Command cmdCommand3 = null;
StringItem strLabelCommand = null;
```

ed in particolare:

Il comando *cmdExit* sarà il comando che (quando selezionato) uscirà dall'applicazione, l'item *strLabelCommand* rappresenta l'item mostrato a video che conterrà la label del comando selezionato (inizialmente non conterrà nulla). I comandi *cmdCommand1*, *cmdCommand2*, *cmdCommand3* una volta selezionati cambieranno il valore dell' item *strLabelCommand*.

Il costruttore della MIDlet

```
public EsempioMidlet() {
   super();
   cmdExit = new Command("Esci", Command.EXIT, 1);
   cmdCommand1 = new Command("Command1", Command.SCREEN, 1);
   cmdCommand2 = new Command("Command2", Command.SCREEN, 1);
   cmdCommand3 = new Command("Command3", Command.SCREEN, 1);
}
```

non fa altro che inizializzare i comandi. In particolare è da notare come il comando cmdExit sia stato creato con un tipo differente dagli altri. Il motivo è da ricercare nel fatto che al comando cmdExit si sta assegnando un significato particolare: quello di uscita dall'applicazione e quindi il tipo EXIT è quello che fa al caso nostro. Gli atri comandi hanno un significato solo all'interno di questa applicazione (anzi all'interno di questo schermo) quindi il tipo SCREEN è più che adatto. La priorità dei comandi non è importante, quindi sono state lasciate tutte ad uno.

I metodi pauseApp() e destroyApp() non sono stati implementati in quanto non significativi per questo esempio, mentre il metodo startApp() fa diverse cose importanti:

```
protected void startApp() {
   Form frmMain = new Form("Esempio");
   frmMain.addCommand(cmdExit);
   frmMain.addCommand(cmdCommand1);
   frmMain.addCommand(cmdCommand2);
   frmMain.addCommand(cmdCommand3);
   frmMain.setCommandListener(this);
   strLabelCommand = new StringItem("Label Command", "");
   frmMain.append(strLabelCommand);
   Display.getDisplay(this).setCurrent(frmMain);
 }
```

- **Costruisce un oggetto di tipo Form che rappresenta lo schermo dell'applicazione**
- **Aggiunge allo schermo i comandi**
- **Imposta la midlet (l'oggetto this) come ascoltatore dei comandi della form (in una progrogetto reale magari il listener sarebbe stato una classe a parte più specializzata)**
- **Crea un oggetto di tipo StringItem e lo aggiunge alla form. Questo oggetto conterrà le label dei comandi selezionati**
- **Aggiunge l'item appena costruite alla form**
- **Imposta come schermo corrente della MIDlet la form appena configurata.**

Quando la MIDlet viene lanciata appare uno schermo simile al seguente:

Premendo uno dei tue tasti associati ai comandi Esci e Menuviene invocato il metodo commandAction della MIDlet

```
public void commandAction(Command cmd, Displayable disp) {

   if (cmd == cmdExit) {
      destroyApp(false); notifyDestroyed();
    } else {
      String label = cmd.getLabel();
      strLabelCommand.setText(label);
    }
}
```

- **Il metodo prende in input due parametri :**
- **Il comando dal quale proviene l'evento**
- **Il display al quale il comando appartiene**

L'implementazione del metodo è molto semplice. Se il comando selezionato è cmdExit allora viene eseguita una procedura di terminazione della MIDlet, altrimenti (indipedentemente da quale comando è stato selezionato) viene presa la label del comando e impostata come testo dell' item. La scelta di non discriminare ulteriormente su quale comando è stato selezionato dipende dal fatto che la form ha solo quattro comandi. Uno è quello di uscita e gli altri tre sono quelli funzionali.

Da notare che in basso a destra dello schermo del terminale compare il comando Menu. Questo comando non è stato implementato da noi, ma dal terminale. Infatti avendo creato tre comandi di tipo SCREEN e non avendo tasti a sufficienza per tutti i comandi dello schermo, l'implementazione del MIDP ha preferito creare dinamicamente un comando ed assegnare a questo comando un menu di comandi. Selezionando il comando Menu lo schermo diventa :

Da questo schermo (inventato e gestito interamente dal sistema) è possibile ripremere il tasto Menu per tornare allo schermo precedente o selezionare un comando. Nel caso in cui si selezionasse il comando Command2, lo schermo diventerebbe :


Esempio
Label Command Command2
Esci
Menu