

Java per Dispositivi Mobili: La Piattaforma J2ME

Stefano Valle – Luca Vassena

28/05/2004

Scaletta Generale

- Storia J2ME
- J2ME in dettaglio: Profili, Configurazioni, Virtual Machine
- La struttura di una *MIDlet*
- Package di MIDP con esempi: gestori di schermo, oggetti dell'interfaccia, gestione dati (RMS), I/O
- Nozioni pratiche
- Differenze con J2SE
- Il punto di vista del programmatore
- Conclusioni e sviluppi

2

Storia J2ME

- J2ME: un ritorno alle origini di Java
- Storia:
 - Primi anni '90: Parte OAK del Green Project
Linguaggio OO per dispositivi di consumo (prima idea di *"write once, run anywhere"*)
 - 1996: Java Card
Piattaforma Java per Smart Card
 - 1997: PersonalJava
Per dispositivi connessi alla rete, con interfaccia utente
 - 1998: EmbeddedJava
Per dispositivi Java integrati: ogni costruttore sceglie le API adatte per la sua applicazione
 - 1999: Spotless System e KVM
La più piccola JVM con le funzioni Java di base

3

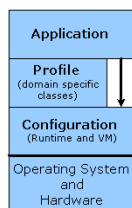
Scaletta in dettaglio

- J2ME in dettaglio:
 - Visione globale
 - Struttura della Piattaforma
 - Le Configurazioni
 - Le Virtual Machines
 - I Profili
 - Packages opzionali

4

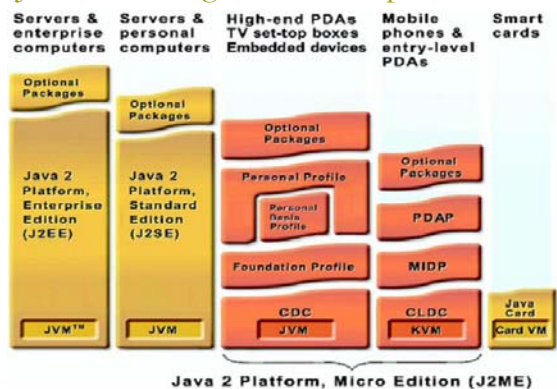
J2ME in dettaglio: visione globale

- Non viene definito un nuovo linguaggio: si "adatta" Java tradizionale alle caratteristiche dei dispositivi mobili
- Struttura di j2me:
 - Configurazioni:
 - CLDC, con relativi profili
 - CDC, con relativi profili



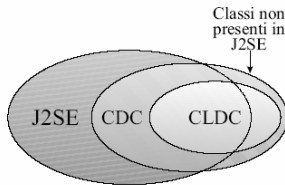
5

J2ME in dettaglio: struttura piattaforma



J2ME in dettaglio: le configurazioni (1/2)

- Una configurazione definisce l'ambiente *runtime* J2ME di base: include la VM ed un set di API fondamentali derivate da J2SE



7

J2ME in dettaglio: le configurazioni (2/2)

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

8

J2ME in dettaglio: le Virtual Machine

- VM per CLDC: **KVM** (K VM)
 - VM compata creata appositamente per dispositivi dalle risorse limitate.
 - Il cuore della KVM occupa 32 - 80 KB.
- VM per CDC: **CVM** (Compact VM)
 - VM con pieno supporto j2se 1.3
 - Più portatile, efficiente, ridotto rispetto all'implementazione standard

9

J2ME in dettaglio: i profili (1/2)

- Un profilo estende la configurazione su cui si basa, aggiungendo classi adatte al dominio di applicazione
- Profili CLDC:
 - MIDP: il primo e più maturo profilo j2me, adatto allo sviluppo di applicazioni per dispositivi wireless come cellulari e smartphone
 - PDAP: profilo creato appositamente per PDA, estende MIDP e CLDC per sfruttare le maggiori capacità di questi dispositivi

10

J2ME in dettaglio: i profili (2/2)

- Profili CDC:
 - FP – Foundation Profile
Profilo di più basso livello, adatto a periferiche integrate senza interfaccia utente (non fornisce infatti supporto GUI). È la base per gli altri profili CDC.
 - PBP – Personal Basis Profile
Estende FP con un sottoinsieme di API AWT per dispositivi con semplici interfacce grafiche.
 - PP – Personal Profile
Estende FP con tutte le API AWT e supporto per Applet. Rappresenta la nuova implementazione di Personal Java.

11

J2ME in dettaglio: packages opzionali

- Java API per Bluetooth
- Wireless Messaging API (WMA)
- Mobile Media API (MMAPI)
- Package RMI
- JDBC per FP
- ...
- Non solo Sun...

12

Scaletta Generale

- Storia J2ME
- J2ME in dettaglio: Profili, Configurazioni, Virtual Machine
- La struttura di una *MIDlet*
- Package di MIDP con esempi: gestori di schermo, oggetti dell'interfaccia, gestione dati (RMS), I/O
- Nozioni pratiche
- Differenze con J2SE
- Il punto di vista del programmatore
- Conclusioni e sviluppi

13

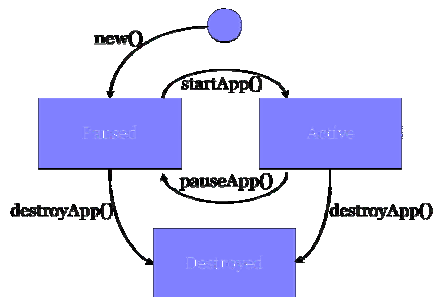
La struttura di una MIDlet (1/2)

- Un'applicazione definita nel profilo MIDP viene chiamata *MIDlet*
- Il ciclo di vita di una applicazione (caricamento – esecuzione – distruzione) è simile al modello utilizzato nelle *Applet*

14

La struttura di una MIDlet (2/2)

- Ciclo di vita di una *MIDlet*



15

Scaletta Generale

- Storia J2ME
- J2ME in dettaglio: Profili, Configurazioni, Virtual Machine
- La struttura di una *MIDlet*
- Package di MIDP con esempi: gestori di schermo, oggetti dell'interfaccia, gestione dati (RMS), I/O
- Nozioni pratiche
- Differenze con J2SE
- Il punto di vista del programmatore
- Conclusioni e sviluppi

16

Scaletta in dettaglio

- I package di MIDP 1.0
- Package `lcdgui`
 - Gerarchia, `Display` & `Displayable`, `Canvas`, `Form` & `Item`, `List`, `Alert`, `Command` & `CommandListener`
- Package `rms`
- Package `microedition.io`

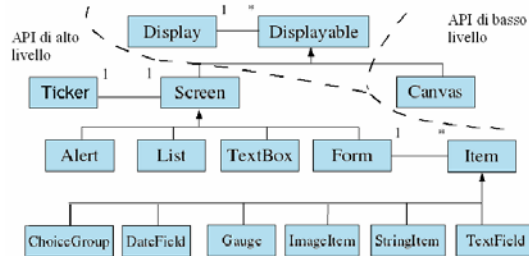
17

I package di MIDP 1.0

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

18

javax.microedition.lcdui: gerarchia



19

Display & Displayable

Display

- oggetto rappresentante il display fisico del dispositivo
- unico e vi si accede tramite pattern Singleton
- metodi:
 - `setCurrent(Displayable)`, `getCurrent()`
 - `isColor()`, `getNumColors()`, ...

Displayable

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

20

Primo esempio (1/2)



21

Primo esempio (2/2)

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

public class PrimoEsempio extends MIDlet {
    private Display schermo;
    private Displayable schermata;

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

    public void pauseApp() {
    }

    public void startApp() {
        schermo = Display.getDisplay(this);
        schermata = new Form("Sono una form");
        schermo.setCurrent(schermata);
    }
}
```

22

Commenti (1/3)

Ogni applicazione importa il package `javax.microedition.midlet`! Gli altri package a seconda delle necessità.

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

La classe base dell'applicazione deve ereditare `javax.microedition.MIDlet`

```
public class PrimoEsempio extends MIDlet {
```

`Display` rappresenta lo schermo, `Form` è un tipo di schermata

```
private Display schermo;
private Displayable schermata;
```

23

Commenti (2/3)

Metodi per la gestione dell'applicazione.

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

public void pauseApp() {}

public void startApp() {...}
```

Ottingo il display.

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

Singleton

Creo una schermata.

```
schermata = new Form("Sono una form");
```

24

Commenti (3/3)

Mostro la schermata sul display.

```
schermo.setCurrent(schermata);
```



25

Canvas (1/2)

- Programmatore ha libertà totale, ma ciò implica maggiore complessità
- Pittura come in J2SE: metodo `paint` e oggetto `Graphics`
- Unica schermata per gestire autonomamente eventi dai tasti e dal puntatore
- In J2SE per gestire un evento devo implementare l'apposita interfaccia e riempire i metodi; qui devo solo riempire i metodi appositi (sono astratti nella classe `Canvas`)
- `Canvas` è l'unico a ricevere gli eventi: non posso creare oggetti con propri Listener
- Mai dipingere in coordinate assolute, ma sempre relative (troppi dispositivi con schermi troppo differenti!!!)

26

Canvas (2/2)

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

class MyCanvas extends Canvas{
    private int h = getHeight();
    private int w = getWidth();

    public MyCanvas(){

    }

    protected void paint(Graphics g){
        g.setColor(255, 255, 255);
        g.fillRect(0, 0, w, h);
        g.setColor(0, 255, 0);
        g.drawString("W PAOO", w/2, h/2,
                    Graphics.TOP | Graphics.HCENTER);
    }

    protected void KeyPressed(int keyCode){}

    protected void pointerPressed(int x, int y){}
}
```

pittura come in J2SE

Coordinate relative

alcuni metodi per gestire eventi



Form & Item (1/8)

- `Form`: sottoclasse di `Screen` che è sottoclasse di `Displayable`
- Concetto simile a quello di `Frame` o `Window` nelle AWT
- Al contrario di `Canvas` è totalmente gestita dal sistema (presentazione, scroll)
- `Form`: unica schermata in cui si possono inserire gli `Item`

28

Form & Item (2/8)

- `Item`: oggetti per l'inserimento di dati (Component di J2SE)
- Al contrario di J2SE non puoi estendere `Item` per creare propri componenti (in MIDP 1.0)
- Ecco gli `Item`:
 - `TextField`: inserimento di stringhe caratteri, analogo J2SE
 - `ChoiceGroup`: equivale a `RadioButton` e `CheckBox` di J2SE



29

Form & Item (3/8)

- `StringItem`: stringa con etichetta (Label)
- `ImageItem`: immagine con titolo e layout



30

Form & Item (4/8)

- **DateField**: inserimento di data e ora
- **Gauge**: barra di riempimento (editabile e non), simile J2SE



31

Form & Item (5/8)

Estendo la classe **Form** per creare la mia form, dichiaro i componenti che utilizzerò

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

public class MyForm extends Form {
    private TextField testo;
    private ChoiceGroup scelta;
    private StringItem stringa;
    private ImageItem immagine;
    private DateField data;
    private Gauge barra;

    public MyForm() {
        super("Titolo");
    }
}
```

32

Form & Item (6/8)

Creiamo il campo di testo, passandogli etichetta, testo, limite di caratteri, vincolo

```
testo = new TextField("TextField", "testo", 10, TextField.ANY);
```

Attacchiamolo alla Form con il metodo **append(Item)** di **Form**. Analogo al metodo **add** dell'oggetto **Container** nelle **AWT**

```
this.append(testo);
```

Creiamo il **ChoiceGroup**, passandogli etichetta e definendo il tipo (esclusivo o no)

```
scelta = new ChoiceGroup("Choice", Choice.EXCLUSIVE);
scelta.append("Uno", null);
scelta.append("Due", null);
this.append(scelta);
```

Creiamo lo **StringItem** passandogli etichetta e testo

```
stringa = new StringItem("String: ", "stringa");
this.append(stringa);
```

33

Form & Item (7/8)

Creiamo l'**ImageItem**, passandogli etichetta, immagine, layout e testo alternativo

```
try {
    immagine = new ImageItem("Image", Image.createImage("./img.png"),
        ImageItem.LAYOUT_CENTER, "Testo alternativo");
} catch (Exception e) {}
this.append(immagine);
```

Creiamo un **DateField**, passandogli etichetta ed il tipo (data e ora)

```
data = new DateField("Date", DateField.DATE_TIME);
this.append(data);
```

Creiamo un **Gauge**, passandogli etichetta, un booleano per dire se è interattivo, il valore massimo e il valore iniziale

```
barra = new Gauge("Gauge", false, 10, 5);
this.append(barra);
```

34

Form & Item (8/8)

Il risultato:



35

List

- È il classico menu del telefonino
- È identica ad una Form di soli **ChoiceGroup** esclusivi
- È composta da un insieme di stringhe e/o immagini
- Come menu di J2SE, ma più semplificato

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

public class MyList extends List {

    public MyList() {
        super("Sono una List", List.IMPLICIT);
        this.append("Uno", null);
        this.append("Due", null);
        this.append("Tre", null);
        this.append("Quattro", null);
    }
}
```

Appendo stringhe e non immagini



36

Alert

- Tipica schermata di errore o di avviso
- Visualizza una finestra di dialogo a
 - tempo determinato: numero di millisecondi specificato dal programmatore
 - tempo indeterminato: finché utente non clicca su "ok"

```
Image image = Image.createImage("/immagine.png");
Alert alert = new Alert("Label", "Testo", image, AlertType.ERROR);
alert.setTimeout(3000);

Display.getDisplay(MyMidlet.instance).setCurrent(alert, this);
```

Mostra per 3 secondi Alert e poi passa all'altra schermata indicata; in questo caso torna alla schermata da cui è partito l'Alert

37

Command & CommandListener (1/5)

Command

- È un bottone e viene posizionato sempre nella parte bassa della schermata
- Se il numero di Command è minore di 2, vengono posti negli angoli inferiori della schermata; altrimenti quello a priorità maggiore viene visualizzato singolarmente e gli altri vengono richiamati da un menu
- Nei cellulari vengono richiamati dai "soft buttons"
- Contiene solo informazioni sul comando (priorità, tipo), ma non sa nulla sull'azione da compiere

CommandListener

- Principale ascoltatore degli eventi in una schermata
- Analogo ai Listener di J2SE



Command e soft buttons

Command & CommandListener (2/5)

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

public class MyMidlet extends MIDlet {

    public MyMidlet() {}

    public void pauseApp() {}

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

    public void startApp() {
        Display.getDisplay(this).setCurrent(new MyList2(this));
    }
}
```

Classe base per l'applicazione

39

Command & CommandListener (3/5)

```
import javax.microedition.lcdui.*;
public class MyList2 extends List {
    private MyList2 istanza;
    private MyMidlet midlet;

    public MyList2(MyMidlet m) {
        super("Sono una List", List.IMPLICIT);
        istanza = this;
        midlet = m;
        this.append("Uno", null);
        this.append("Due", null);
        this.append("Tre", null);
        Command exit = new Command("Exit", Command.EXIT, 1);
        this.addCommand(exit);
        this.setCommandListener(new MyList2Listener());
    }

    class MyList2Listener implements CommandListener {
        public void commandAction(Command cmd, Displayable disp) {
            if (cmd.getCommandType() == Command.EXIT)
                midlet.destroyApp(true);
            else { //selezionato da lista
                int n = istanza.getSelectedIndex();
                Display.getDisplay(midlet).setCurrent(new Form("Selec "+n));
            }
        }
    }
}
```

Schermata e suo Listener

Command & CommandListener (4/5)

Il CommandListener deve sempre implementare il metodo CommandAction. Esso riceve due parametri: comando che ha generato l'evento e la schermata che lo contiene

```
class MyList2Listener implements CommandListener {
    public void commandAction(Command cmd, Displayable disp) {
```

Se il comando è di tipo EXIT richiamo il metodo per chiudere l'applicazione, altrimenti significa che ho selezionato un elemento della lista: in questo caso individuo l'elemento selezionato e mostro una nuova schermata

```
if (cmd.getCommandType() == Command.EXIT)
    MyMidlet.instance.destroyApp(true);
else { //selezionato da lista
    int n = istanza.getSelectedIndex();
    Display.getDisplay(MyMidlet.instance).setCurrent(new Form("Selec "+n));
}
```

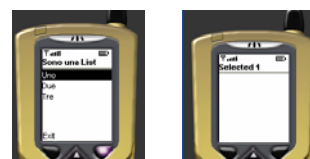
41

Command & CommandListener (5/5)

- Creiamo il Command, passandogli etichetta, tipo e priorità
- Attacciamo Command alla schermata
- Definiamo il CommandListener

```
Command exit = new Command("Exit", Command.EXIT, 1);
this.addCommand(exit);
this.setCommandListener(new MyList2Listener());
```

Il risultato:

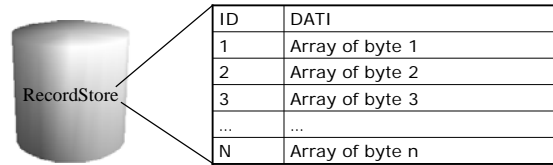


42

javax.microedition.rms (1/2)

- RMS (Record Management System): sistema per la memorizzazione permanente di dati in una struttura lontanamente simile ad un DB relazionale.
- Classi
 - RecordStore: "contenitore" in cui vengono memorizzati i dati. È assimilabile al concetto di tabella nel DB relazionale. Presenta due soli campi:
 - *Campo chiave*: valore numerico fungente da chiave primaria, gestito totalmente dal sistema
 - *Campo dati*: contenente dati sottoforma di array di byte
- Interfacce:
 - RecordComparator: per comparare i record tra loro
 - RecordFilter: per selezionare record particolari
 - RecordEnumerator: per enumerare i record
 - RecordListener: per gestire cambiamenti nei record

javax.microedition.rms (2/2)



44

javax.microedition.io

J2SE: una classe per ogni protocollo
J2ME: classe generica Connector, poi da specializzare.

```
private StreamConnection s = null;  
private OutputStream out = null;  
private InputStream in = null;  
  
try {  
    s = (StreamConnection)Connector.open("comm:2; baudrate=9600;",  
                                         Connector.READ_WRITE);  
    in = s.openInputStream();  
    out = s.openOutputStream();  
} catch (Exception e) {}
```

Protocollo	Parametro per Connector.open()	Tipo di connessione
HTTP	http://java.sun.com	HttpConnection
Socket	socket://4444	StreamConnection
Datagram	datagram://127.0.0.1	DatagramConnection
Seriale	comm:0;baudrate=2400	StreamConnection

45

Scaletta Generale

- Storia J2ME
- J2ME in dettaglio: Profili, Configurazioni, Virtual Machine
- La struttura di una MIDlet
- Package di MIDP con esempi: gestori di schermo, oggetti dell'interfaccia, gestione dati (RMS), I/O
- Nozioni pratiche
- Differenze con J2SE
- Il punto di vista del programmatore
- Conclusioni e sviluppi

46

Scaletta in dettaglio

- Nozioni pratiche
 - I file importanti
 - Wireless Toolkit
 - Installazione delle applicazioni
 - Esecuzione delle applicazioni

47

I file importanti (1/2)

- JAR: archivio contenente tutte le risorse, quali classi, immagini e il file manifest
- MANIFEST: contiene metainformazioni sull'applicazione (dimensione, profilo ecc...), va inserito nel JAR
- JAD: descrittore del file JAR
 - Permette di avere le informazioni base sull'applicazione prima di scaricarla (per non scaricare applicazioni di un profilo diverso da quello supportato)
 - Permette di passare parametri all'applicazione senza dover mettere mano ai sorgenti

48

I file importanti (2/2)

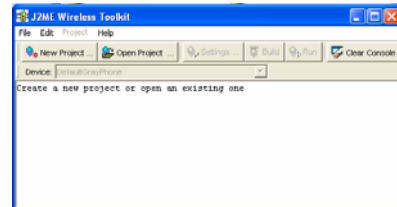
- Campi del file JAD

Nome attributo	Funzione	Richiesto
MIDlet-Name	Nome della MIDlet	Sì
MIDlet-Version	Numero di versione	Sì
MIDlet-Vendor	Autore	Sì
MIDlet	Classe principale	Sì
MicroEdition-Profile	Profilo J2ME	Sì (MIDP)
MicroEdition-Configuration	Configurazione J2ME	Sì (CLDC)
MIDlet-Icon	Icona dell'applicazione	No
MIDlet-Info-URL	URL per info supplementari	No

49

Wireless toolkit (1/2)

- Installare un JDK o JRE 1.3 o superiore
- Recuperare dal sito della Sun il J2ME Wireless Toolkit (l'ultimo è il 2.0)
- Installare il Wireless Toolkit
- "New project" per una nuova applicazione



50

Wireless toolkit (2/2)

- Vengono chieste informazioni e poi viene creata la cartella per l'applicazione: `.../cartellaWT/apps/mioProgetto`
- La struttura è la seguente:
 - bin: cartella per JAR e JAD
 - res: cartella per le risorse (immagini)
 - src: cartella per i sorgenti
 - classes: cartella per le classi compilate
- Scrivete il vostro codice, mettete i sorgenti nella cartella res e poi "Build" per compilare e "Run" per eseguire

51

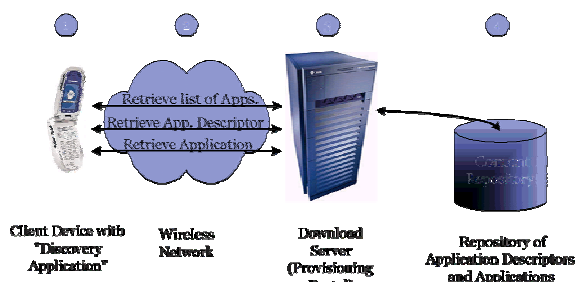
Installazione delle applicazioni (1/2)

- Per installare la propria applicazione su cellulare supportante java basta copiarla nel dispositivo
- Esistono due modi per farlo
 - Utilizzare il cavetto dati del cellulare
 - Utilizzare un metodo di approvvigionamento: OTA

52

Installazione delle applicazioni (2/2)

- Metodo di approvvigionamento OTA



53

Esecuzione delle applicazioni su PDA

- Per eseguire la propria applicazione su PDA:
 - Installare una virtual machine per MIDP
 - IBM J9 (parte del WebSphere Studio)
 - JBed
 - Installare una virtual machine Personal Java e le librerie ME4SE (me4se.org)
 - Jeode
 - CrEme
 - Sun Personal Java
 - ChaiVM

Per cellulare J2ME è la soluzione migliore (portabilità, "write once, run everywhere")
Per PDA: c'è anche .NET, eweSoft ecc...

54

Scaletta Generale

- Storia J2ME
- J2ME in dettaglio: Profili, Configurazioni, Virtual Machine
- La struttura di una *MIDlet*
- Package di MIDP con esempi: gestori di schermo, oggetti dell'interfaccia, gestione dati (RMS), I/O
- Nozioni pratiche
- Differenze con J2SE
- Il punto di vista del programmatore
- Conclusioni e sviluppi

55

Differenze con J2SE (1/2)

- Limitazioni di J2ME rispetto a J2SE:
 - *Calcoli in virgola mobile*
 - Fanno un uso intenso del processore
 - Qui non c'è co-processore matematico
 - Assenti i tipi *float* e *double*, non supportati dal linguaggio
 - *Finalizzazione*
 - Non è possibile dichiarare un metodo *finalize()*
 - L'overload sarebbe troppo alto in termini di risorse
 - *Gruppi di thread e thread demoni*
 - Non esiste la classe *ThreadGroup*
 - I thread sono trattati singolarmente

56

Differenze con J2SE (2/2)

- *Gestione delle eccezioni*
 - L'insieme degli errori inclusi in CLDC è limitato: capacità ristretta di gestire eccezioni
 - Gestione in J2SE molto complessa, è difficile da realizzare con processori limitati
 - In sistemi embedded il recupero da condizioni di errore è specifico del dispositivo
- *Interfaccia nativa Java (JNI) non implementata*
 - Implementazione troppo costosa
- *Non dispone della Reflection*

57

Il punto di vista del programmatore (1/2)

- ... nella scrittura del codice:
 - Attenzione a tutte le caratteristiche non supportate da J2ME (floating point, ...)
 - Scrivere codice più compatto
 - Ottimizzazione del codice nella compilazione (es. con JBuilder)

58

Il punto di vista del programmatore (2/2)

- ... nella progettazione:
 - Focalizzarsi su una categoria di dispositivi target
 - Gestire risorse limitate
 - Lo schermo è piccolo
 - Inserire solo immagini piccole ed ottimizzate
 - Diagrammi dell'interfaccia
 - UML non basta...
 - Nascono sistemi nuovi di progettazione grafica dell'interfaccia

59

Diagrammi dell'interfaccia (1/2)

- Il punto di vista cambia:

UML

Le classi
composizione
e relazioni fra
di esse

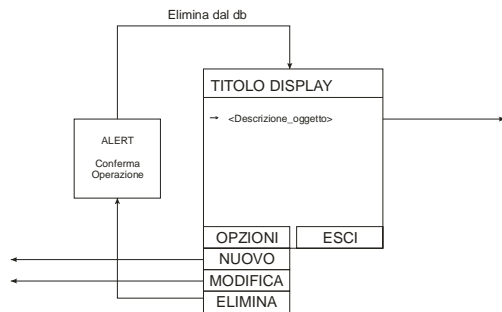


DIAGRAMMI

L'interfaccia grafica
disposizione degli
oggetti nello schermo
ed interazione fra le
finestre

60

Diagrammi dell'interfaccia (1/2)



61

Sviluppi (1/2)

■ MIDP 2.0

- Connessioni I/O: supporto HTTPS, sockets, datagrams, porte seriali, IrDA
- Gestione avanzata delle form: *CustomItem*
- Supporto multimediale avanzato (parte di MMAPi)
- Sicurezza: concetti di *permesso*, *codice sicuro* / *non sicuro*, *dominio di protezione*

62

Sviluppi (2/2)

- API dedicata allo sviluppo di giochi
- Supporto OTA integrato
- Gestione avanzata RMS (possibilità di condividere i dati tra applicazioni)
- Caratteristiche avanzate: *Push Registry*, *Platform Request*

63

Conclusioni

- *Write once, run anywhere*: in realtà non è più così vero...
 - Dispositivi troppo differenti
- Ogni costruttore fa quello che vuole
- Come tecnologia è promettente, ma è ancora giovane...

64

Bibliografia

- Articoli tecnici dal sito Sun (java.sun.com)
- Tesi varie...
 - Stefano Valle
Controllo e Gestione di Sistemi Domotici: Interazione con Comunicatori GSM Tramite Cellulari Java
 - Luca Vassena
Controllo e gestione di dispositivi domotici: interruttore multifunzione e PDA
 - Paolo Zuliani
Controllo e Gestione di Sistemi Domotici: PDA e Comunicazione Bluetooth con Sistemi PowerLine e Bus Domotici
- Minicorsi: www.wmlscript.it
- Esistono molti libri su J2ME...

65