J2ME Avanzato

Antonio Tereno
http://java2me.org/

Pianificazione dei compiti

- La classe `Timer` pianifica l'esecuzione di un compito
- La classe `TimerTask` esegue un compito

- Il timerTask e' il compito che verrà eseguito, timer specifica quando (con che ritardo, ripetizione, in che data)

Timer: Schedulazione

- `schedule(TimerTask task, long delay)`
  - Compito eseguito dopo un ritardo delay
- `schedule(TimerTask task, long delay, long period)`
  - Compito a ritardo fisso (delay)
- `scheduleAtFixedRate`
  `(TimerTask task, long delay, long period)`
  - Compito a ripetizione fissa
- `schedule(TimerTask task, Date time)`
  - Compito eseguito dopo un ritardo delay
- `schedule(TimerTask task, Date firstTime, long period)`
  - Compito a ritardo fisso (delay)
- `scheduleAtFixedRate`
  `(TimerTask task, Date firstTime, long period)`
  - Compito a ripetizione fissa

Il task

Andremo ad implementare il metodo astratto run():

```
Timer tm = new Timer();
NostroTaks tt = new NostroTask();
tm.schedule(tt,10000);

private class NostroTask extends TimerTaks{
    public final void run(){
        System.out.println(
                “Questa scritta viene “ +
                ”stampata ogni 10 secondi”
        );
    }
}
```

RMS, Record Management System

- Il sistema adottato è un semplice database orientato al record che è, quindi, l'unità minima di informazione manipolabile.
- Qualunque operazione sui dati (scrittura, lettura, modifica e cancellazione) sarà effettuata sui singoli record.

RMS, Record Management

- L'insieme dei record forma un record store che può essere paragonato alla tabella di un database e che rappresenta quindi "il più alto grado di aggregazione dei dati" (non esiste un oggetto predefinito che raggruppi più record store (tabelle)

I Record Store

- Per l'apertura e/o creazione di un record store occorre richiamare il metodo (statico) della classe

  `RecordStore (javax.microedition.rms )`

- ```
  public static RecordStore
     openRecordStore
      (
           String recordStoreName,
           boolean createIfNecessary
     );
  ```
  ```

RMS: gestione degli errori

- `RecordStoreException`
  - per segnalare una generica "condizione anomala"
- `RecordStoreFullException`
  - per indicare il riempimento del record store
- `RecordStoreNotFoundException`
  - ad indicare che il record store richiesto (recordStoreName) non esiste o comunque è inaccessibile

Chiusura e di Distruzione

- `public void closeRecordStore();`
- `public static RecordStore deleteRecordStore(String rsName);`

- Possono alzare le seguenti exception:

- `RecordStoreException`
- `RecordStoreNotOpenException (II)`
- `RecordStoreNotFoundException`

RMS: l'Header

- Tutte le informazioni contenute nell'header sono accessibili tramite opportuni metodi:

- ```
  public long getLastModified();
  public int getNextRecordId();
  public int getNumRecords();
  public int getVersion();
  ```
  ```

Altri metodi Sul RS

- `public int getSizeAvailable();`
- `public int getSize();`
- `public String[] listRecordStores();`

Add/Delete

- `public int addRecord`
  `( byte[] data,`
  `int offset,`
  `int byteNr);`
- `public void deleteRecord`
  `(int recordId);`

Get/Set

- `public byte[] getRecord(int recordId);`
- `public int getRecord`
  `( int recordId,`
  `byte[] buffer,`
  `int offset);`
- `public void setRecord`
  `(int recordId, byte[] data, int offset, int byteNr);`

RecordListener

- `public void recordAdded (RecordStore store, int recordId);`
- `public void recordDeleted (RecordStore store, int recordId);`
- `public void recordChanged (RecordStore store, int recordId);`

RecordListener

- `public void addRecordListener (RecordListener listener);`

- ogni operazione sullo store genererà una chiamata ad uno dei tre metodi appena visti in base all'operazione effettuata (aggiunta - cancellazione - modifica).

RecordEnumeration

- ```
  for(int i=1; i<= store.getNumRecords(); i++)
  {
    byte[] record = store.getRecord(i);
  }
  ```
- A seguito di operazioni di cancellazione fatte sul record store (deleteRecord (int recordId), può accadere che alcuni id passati al metodo getRecord(i) nel ciclo for non corrispondano ad alcun record con un conseguente lancio dell'eccezione `InvalidRecordIdException`
  ```

Enumeration

- per ottenere una enumerazione di record occorre richiamare il metodo della classe RecordStore :
- ```
  RecordEnumeration
  enumerateRecord
  (
    RecordFilter filter,
    RecordComparator comparator,
    boolean keepUpdated
  );
  ```
  ```

“Muoversi” tra i records

- I metodi:

```
public boolean hasNextElement();
public boolean hasPreviousElement();
```

- verificano la presenza o meno di un record "nelle due direzioni", mentre con i metodi:

```
public byte[] nextRecord();
public byte[] previousRecord();
public int nextRecordId();
public int previousRecordId();
```

- è possibile ottenere il record precedente o quello successivo o i loro indici.

Reset/Destroy

- Un oggetto di tipo RecordEnumeration può essere resettato (`public void reset()`), in questo caso tutti gli indici dell'enumerazione vengono riportati allo stato in cui si trovavano quando l'"enumeratore" è stato creato, chiamando invece il metodo:
- `public void destroy();`
- vengono liberate tutte le risorse di cui il `RecordEnumeration` disponeva.

Filter/Compare

- Le interfacce `RecordFilter` e `RecordComparator` forniscono due semplici strumenti per il "filtraggio" e l'ordinamento dei record.

- L'interfaccia `RecordFilter` possiede un solo metodo:

```
public boolean matches(byte[] candidateRecord);
```

RecordComparator

```
public int compare(byte[] record_1, byte[] record_2);
```

- Come nel caso del RecordFilter, viene richiamato da RecordEnumeration. In questo metodo l'enumerazione tiene conto del risultato dell'operazione di "confronto" (il criterio di confronto deve essere implementato dal programmatore). In particolare tale risultato può essere:

- `EQUIVALENT`: se, per il criterio di ricerca o di ordinamento, i due record sono uguali;
- `FOLLOWS`: se il primo record (record_1), in base ai criteri di ricerca o di ordinamento, segue record_2;
- `PRECEDES`: se il primo record (record_1), in base ai criteri di ricerca o di ordinamento, precede record_2;

l'interfaccia RecordEnumeration

```
RecordEnumeration e =
store.enumerateRecords(null,null,true
);
while(e.hasNextElement()){
  byte[] record = e.nextRecord();
}
```

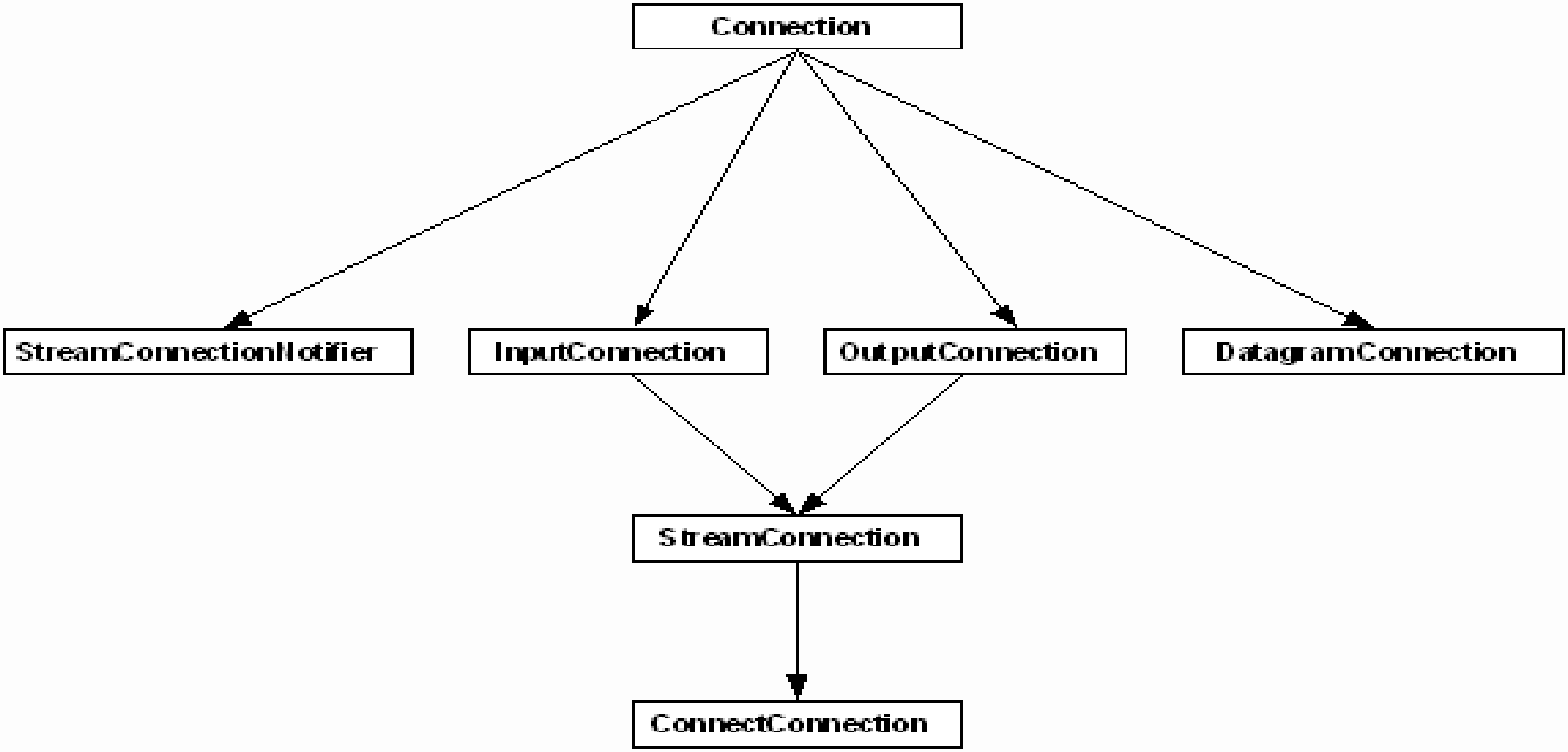
Generic Connection Framework

- Si dispone della classe Connector, con cui si può creare ogni tipo di connessione:

```
Connector.Open(“protocol:address;parameters”);
```

```
Protocol    := http | socket | file
Address     := www.google.it | 127.0.0.1 | testFile.txt
Parameters  := 8080
```

Gerarchia GCF



Connessione

- L'unico protocollo di cui e' garantita l'implementazione in MIDP 1.0...

- Per aprire una connessione:

```
ContentConnection c =
   (ContentConnection)Connector.Open(url
     );
```

GCF improved in MIDP 2.0!

- Https
- Thread separato: tutte le chiamate a metodi che coinvolgono networking

Il Wireless Toolkit

- WTK 1.04 MIDP1
- WTK 2.1
  - MIDP2
  - MMA 1.1
  - WMA 1.1
  - WSA 1.0

http://java.sun.com/products/j2mewtoolkit/

Usare Ant con J2ME

Ant è un tool cross platform OpenSource per la compilazione

Ant:java = make:C!

http://ant.apache.org

Ant: build.xml

```
<project name="Hello" default="clean"
   basedir=".">
<property name="program_name" value="Hello"/>
<property name="package_name" value="Hello"/>
<property name="proguard“
   value=“c:/proguard/lib/proguard.jar"/>
<!–- il path dell’offuscatore proguard -->
   <property name="midp_home"
   value="e:/j2mewtk"/>
<!-- il path completo del WTK ->
   <property name="midp_lib"
   value="${midp_home}/lib/midpapi.zip"/>
<!-- il path completo delle MIDP Api ->
```

Build.xml: le properties

```
<property name="top" value="."/>
<property name="src" value="src"/>
<!-- la directory dei sorgenti -->
<property name="res" value="res"/>
<!- la directory delle risorse -->
<property name="build" value="build"/>
<!- la directory per il build (automatica) -->
<property name="obf" value="obf"/>
<!- la directory delle classi offuscate (autom.)-->
<property name="deploy“ value="deploy“>
<!- la directory risultato (autom.) -->
```

Build.xml: clean&init

```
<target name="clean">
<delete dir="${build}"/>
<delete dir="${deploy}"/>
<delete dir="${obf}"/>
</target>
<!– Cancella le vecchie directory ${build}, ${obf} and
${deploy}-->

<target name="init" depends="clean">
<tstamp/>
<mkdir dir="${build}"/>
<mkdir dir="${deploy}"/>
<mkdir dir="${obf}"/>
</target>
<!– Crea le directory necessare-->
```

Ant: compile

```
<target name="compile">
<mkdir dir="${build}/classes"/>
<!- crea la directory classes -->
<javac destdir="build/classes" srcdir="src"
    bootclasspath="${midp_lib}" target="1.1"/>
<!- comando di compilazione -->
</target>
```

Ant: preverify

```
<target name="preverify">
<mkdir dir="${build}/preverified"/>
<exec executable="${midp_home}/bin/preverify">
   <arg line="-classpath ${midp_lib}"/>
   <arg line="-d ${build}/preverified"/>
   <arg line="${build}/classes"/>
</exec>
</target>
<!-- esegue la preverifica delle classi -->
<!-- creando la directory preverified -->
```

Ant: Jaring

```
<target name="dist">
<mkdir dir="${build}/bin"/>
   <jar basedir="${build}/preverified“
      jarfile="${build}/bin/${program_name}.jar“
      manifest="bin/MANIFEST.MF">
      <fileset dir="${top}/${res}">
            <include
   name="${package_name}/*.png"/>
      </fileset>
   </jar>
<copy file="bin/${program_name}.jad“
tofile="${build}/bin/${program_name}.jad"/>
</target>
```

Ant: obfuscate

```
<target name="obfuscate">
<java fork="yes" classname="proguard.ProGuard“
classpath="${proguard}">
<arg line="-libraryjars ${midp_lib}"/>
<arg line="-injars
   ${build}/bin/${program_name}.jar"/>
<arg line="-outjar ${obf}/${program_name}.jar"/>
<arg line="-keep 'public class * extends
javax.microedition.midlet.MIDlet'"/>
</java>
<unjar src="${top}/${obf}/${program_name}.jar“
dest="${top}/${obf}/extract" />
</target>
```

Ant: preverifica

```
<target name="preverifyobf">
<mkdir dir="${build}/preverifiedobf"/>
<exec executable="${midp_home}/bin/preverify">
<arg line="-classpath ${midp_lib}"/>
<arg line="-d ${build}/preverifiedobf"/>
<arg line="${obf}/extract"/>
</exec>
</target>
```

Ant: jar delle classi offuscate

```
<target name="distobf">
<copy file="${build}/bin/${program_name}.jar“
todir="${build}/bin/${program_name}-orig.jar"/>
<mkdir dir="${build}/bin"/>
<jar basedir="${build}/preverifiedobf“
jarfile="${build}/bin/${program_name}.jar“
manifest="bin/MANIFEST.MF">
<fileset dir="${top}/${res}">
<include name="${package_name}/*.png"/>
</fileset>
</jar>
<copy file="bin/${program_name}.jad“
tofile="${build}/bin/${program_name}.jad"/>
</target>
```

Ant: deploy

```
<target name="deploy">
<copy file="${build}/bin/${program_name}.jad“
   tofile="${top}/${deploy}/${program_name}.jad"
   />
<copy file="${build}/bin/${program_name}.jar“
   tofile="${top}/${deploy}/${program_name}.jar"
   />
</target>

<!-- prepara Jad e Jar -->
```

Ant: run

```
<target name="run">
    <exec executable="${midp_home}/bin/emulator">
        <arg line="-Xdescriptor:
                ${build}/bin/${program_name}.jad"/>
    </exec>
</target>

<!-- Esegue l'applicazione con l'emulatore -->
```

Ant: eseguire più task

```
<target name="all-non-obf“
    depends="init,compile,preverify,dist"/>

<target name="all-obf"
    depends="init,compile,preverify,dist,obfuscate,pre
    verifyobf,distobf,deploy,run"/>

<!-- In questo modo potremo copilare, fare il deploy
    ed eseguire con un solo comando! -->
```

MIDP Game API

- GameCanvas
- Sprite
- TiledLayer, per i tiled backgrounds, inclusi gli animated tiles
- LayerManager
- MIDP 2.0 Media API, per I suoni
- Transparent Images
- Full-screen mode
- Vibration
- Backlight flashing

Multimedia API

- MMAPI 1.1
  - Audio
  - Video
- Capture
- Playback

Wireless Messaging Api

- SMS
- CBS
  - Binary e Text
- Funzionano
  - Modalità standard
  - Application 2 Application, su porta

WMA 1.1 & 2.0

- 1.1
  - Dinamic
  - Static Registration
- 2.0
  - MMS
    - Suoni
    - Immagini
    - ...

Bluetooth Api

- Device Management
- Device Discovery
- Service Discovery
- Service Registration
- Communication
  - RFCOMM
  - L2CAP

Per approffondire...

Web Services

- Subset delle Api Java 2 SE :

  XML-Based RPC (JAX-RPC 1.1)

- XML-parsing API basate su un subset di SAX2

Un buon punto da cui iniziare...

Xml parsing con J2ME

- Tre tipi di parser (in generale):
  - Model parser (nanoXml, TinyXML,...)
  - Push parser (ASXMLP, MinMl, ...)
  - Pull parser (kXml, ...)

La mia scelta MIDP1: kXml

- Perché?
  - E’ nato per MIDP
  - E’ stabile e relativamente maturo
  - E’ un pull parser
  - E’ semplice da usare

“Installazione”

Homepage: http://kxml.org/

Ultima versione testata: 2.1.8

Dimensioni: 10.1 Kb

Basta copiare il jar nella directory “lib” della nostra applicazione.

kXml: utilizzo

- Si crea un'istanza di
  `org.kxml.parser.XmlParser`
- Si utilizzano i metodi:
  `skip()` e `read()`
- Per muoversi all'interno del foglio Xml

Un esempio

```
void parse(InputStream is) throws Exception
{
   KXmlParser parser = new KXmlParser();
   parser.setInput(new InputStreamReader(is));
   int eventType = KXmlParser.START_TAG;
   while (eventType != KXmlParser.END_DOCUMENT)
   {
      eventType = parser.next();
      if (eventType == KXmlParser.START_TAG)
      {
```

Esempio di parsing

```
if (parser.getName().equals(“pippo"))
{
      parser.next();
      System.out.print(“Pippo Tag: ”);
      System.out.println(parser.getText());
}
else if (parser.getName().equals(“pluto"))
{
      System.out.print(“Pluto Tag: ”);
      System.out.println(parser.getText());
 } …
```

Il “duro” lavoro del JCP

- RMI – Jsr 66
- JDBC – Jsr 169
- SVG – Jsr 226
- Location Api – Jsr 179
- Obex

http://www.jcp.org/

Q&A

- Se ce ne e' ancora il tempo...

Altrimenti...

http://forum.java2me.org/