Java Sound: Audio digitale dalla teoria alla pratica

Stefano Leone Monni
stefano.monni@gmail.com

Sommario

- L’audio digitale ed il package delle Java Sound API: *javax.sound.sampled*
- Realizzazione di un audio Player in Java
- Realizzazione di un audio Recorder in Java
- Rielaborazione audio tramite tecniche DSP

Java e l'audio digitale: perché il package ***javax.sound.sampled***?

- Il package *javax.sound.sampled* (introdotto nella piattaforma Java 2 a partire dalle JDK 1.3) è il cuore delle Java Sound Api e contiene una collezione di classi ed interfacce utili per la cattura, rielaborazione e riproduzione dell'audio digitale.

- Già da prima dell'introduzione del package *javax.sound.sampled* era comunque possibile riprodurre e registrare file multimediali audio/video tramite le JMF.

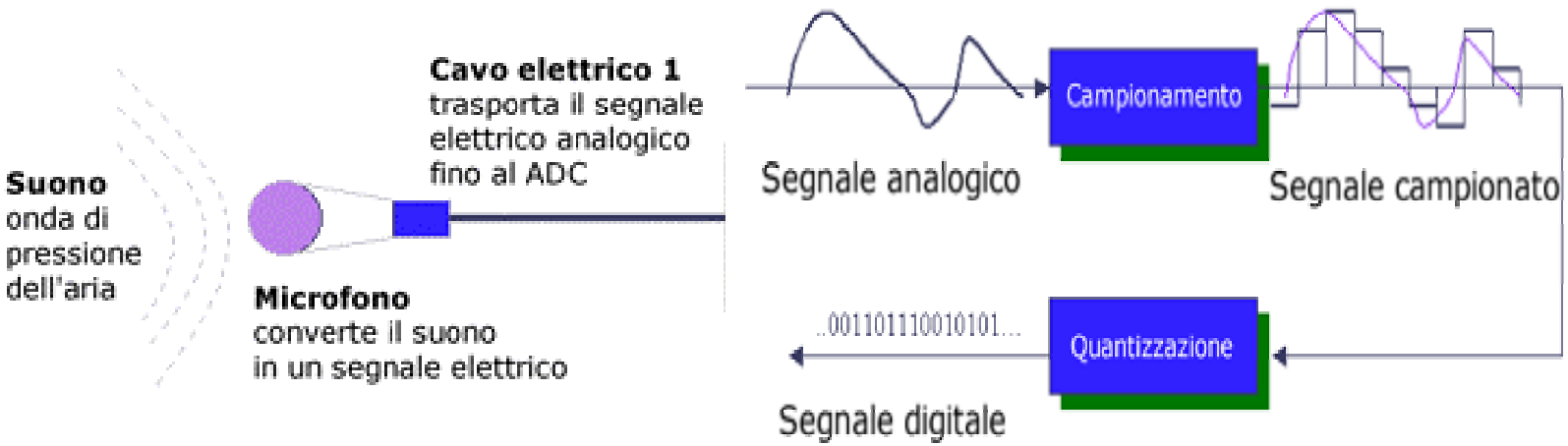
- Perché allora è stato introdotto il package *javax.sound.sampled* ?

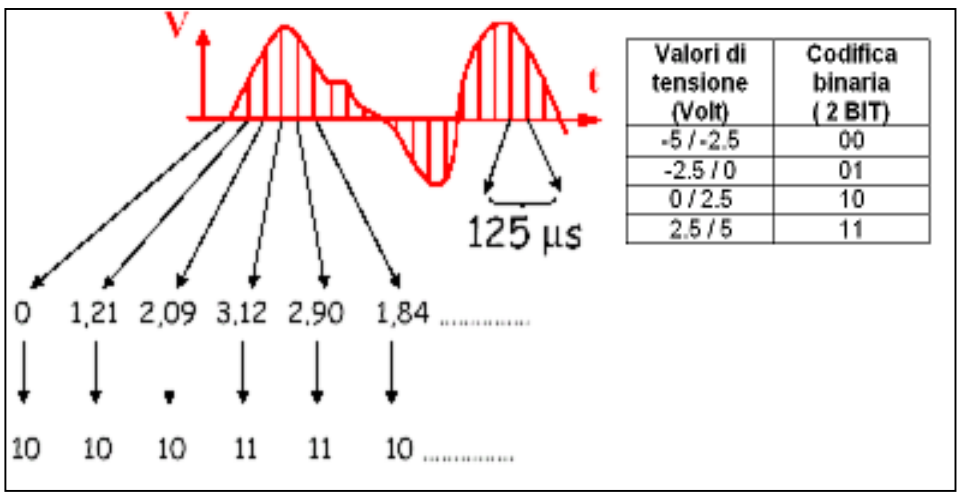
javax.sound.sampled: Gestione a basso livello dell'audio digitale

- Gestione del segnale sonoro a livello di **campione audio**
- Possibilità di applicare tecniche audio DSP per sofisticate elaborazioni del materiale sonoro (amplificazioni, filtraggi, compressioni audio,etc).
- **Ma che cosa si intende per campione audio?**

javax.sound.sampled e l'elaborazione digitale del suono: **i campioni audio** (1/2)

1) **Dall'analogico al digitale**





Valori di tensione (Volt)	Codifica binaria (2 BIT)
-5 / -2.5	00
-2.5 / 0	01
0 / 2.5	10
2.5 / 5	11

→ **Elaborazione digitale del suono (DSP) Byte [ ] samples**

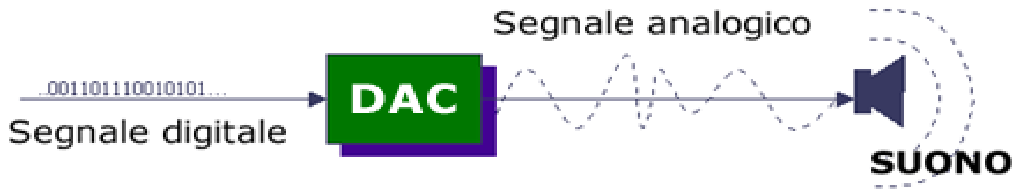
f_c = frequenza di campionamento (Es: 44 kHz)

R = risoluzione audio (quantizzazione) Es: 16 bit

javax.sound.sampled e l'elaborazione digitale del suono: **i campioni audio** (2/2)

2) **Dal digitale all'analogico**

Audio Digitale trasformato tramite DSP



Il package **javax.sound.sampled** consente la gestione dell'audio dalla fase di acquisizione analogica fino a quella di rielaborazione digitale e di riproduzione sonora

Stefano Leone Monni - *stefano.monni@gmail.com*

Realizzazione di un Audio Player in Java

Che cos'è un file audio?

E' un "contenitore" di campioni audio

In genere un file audio contiene una sezione iniziale (header) che specifica il "formato" del file, cioè il modo in cui tali campioni sono in esso rappresentati.

- *numero di byte per campione (Risoluzione audio)*
- *frequenza di campionamento*
- *numero di canali (modalità mono o stereo)*

1) Individuazione dei formati di file audio supportati dal proprio sistema
2) Apertura del file audio selezionato
3) Rilevamento informazioni sul formato audio del file aperto (frequenza di campionamento, risoluzione, etc)
4) Riproduzione sonora del file aperto
5) Intercettazione degli eventi audio

Come rilevare i formati di file audio supportati dal proprio sistema: le classi AudioSystem ed AudioFileFormat

```
AudioFileFormat.Type[] supportedAudioFormat =
AudioSystem.getAudioFileTypes();
```

```
JFileChooser chooser = new JFileChooser(defPath);
ExampleFileFilter filter = new ExampleFileFilter();

int numFormats = supportedAudioFormat.length;


for (int i=0;i<numFormats;i++)
    {filter.addExtension
     (supportedAudioFormat[i].getExtension());
    }

    filter.setDescription("Tutti i file audio
                           supportati");

    chooser.setFileFilter(filter);
    chooser.setDialogTitle("Apertura File audio");
```

Stefano Leone Monni - *stefano.monni@gmail.com*

Come aprire ed ottenere informazioni dal file audio selezionato:
le classi AudioInputStream ed AudioFormat

```
AudioFileFormat aff = null;
AudioInputStream ais = null;
AudioFormat af = null;
...
private boolean openAudioFile (File audioFile)
{
 try{
 aff = AudioSystem.getAudioFileFormat(audioFile);
 ais = AudioSystem.getAudioInputStream(audioFile);
  af = ais.getFormat(); // oppure aff.getFormat()
    } catch (Exception ex)
      {System.out.println(ex); return false; }
 return true;
}
```

Estensione: `aff.getType().getExtension()`

Codifica: `af.getEncoding()`

f_c: `af.getSampleRate()`

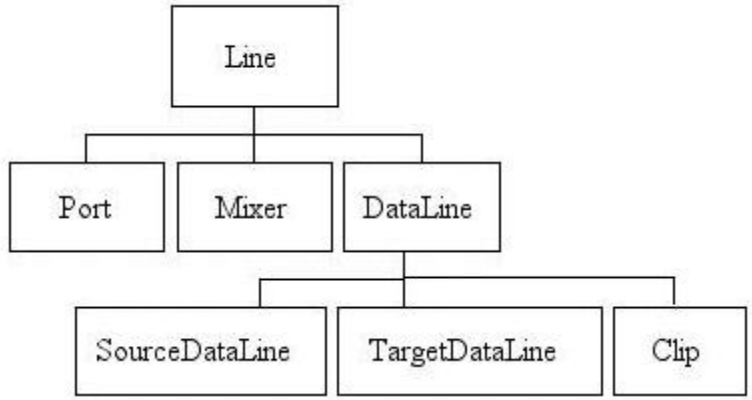
Risoluzione: `af.getSampleSizeInBits()`

Numero di canali: `af.getChannels()`

Stefano Leone Monni - *stefano.monni@gmail.com*

Come ascoltare il file audio aperto: l'impiego degli oggetti DataLine

- Interfacce che consentono ai campioni audio di interagire coi dispositivi I/O di sistema
- Per la riproduzione audio: **SourceDataLine** e **Clip**
- Per la cattura audio: **TargetDataLine**




```
public Clip mySound = null;
...
private boolean
setClipFromAudioInputStream(AudioInputStream ais)
{
  DataLine.Info info =
  new DataLine.Info(Clip.class,
                    ais.getFormat(),
                    ((int) ais.getFrameLength()*
                           af.getFrameSize()));
 try{
     mySound = (Clip) AudioSystem.getLine(info);
     mySound.open(ais);
         ...
     } catch (Exception ex)
        {System.out.println(ex);
          return false;}
```

- per azionare la riproduzione: **mySound.start()**
- per bloccare la riproduzione: **mySound.stop()**
- per spostarsi in una posizione arbitraria del brano audio: **mySound.setFramePosition(nf)**

Come intercettare gli eventi audio: gli ascoltatori LineListener

Descrizione Evento	Metodo	Tipo di Evento
Apertura di una linea	myLine.open()	LineEvent.Type.OPEN
Chiusura di una linea	myLine.close()	LineEvent.Type.CLOSE
Inizio Riproduzione o cattura audio	myLine.start()	LineEvent.Type.START
Fine Riproduzione o cattura audio	myLine.stop()	LineEvent.Type.STOP

```
private LineListener lineListener = new LineHandler();

mySound.addLineListener(lineListener);

public class LineHandler implements LineListener {
  public void update(LineEvent ev) {
    if (ev.getType() == LineEvent.Type.STOP)
    {System.out.println("LineEvent: STOP -> EOF reached  or stop button pressed");
     soundStopped();}
    }}

public void soundStopped();
 {   mySound.setFramePosition(0);
     posSlider.setValue(0);
     playBtn.setSelected(false);
     stopBtn.setSelected(true);}
```

Realizzazione di un Audio Recorder in Java

- Cattura dell'audio da microfono
- Impostazione manuale del formato di registrazione audio (f_c, risoluzione, etc)
- Riproduzione audio delle registrazioni effettuate
- Salvataggio della registrazione sotto forma di file audio con il formato desiderato
- Aggiunta di controlli specifici per la regolazione della riproduzione audio (volume e pan)
- Elaborazione dati audio mediante tecniche DSP

Impostazione manuale del formato audio di registrazione: impiego della classe AudioFormat


```
AudioFormat tmpAf =
            new AudioFormat(AudioFormat.Encoding encoding,
                            float sampleRate,
                            int sampleSizeInBits,
                            int channels,
                            int frameSize,
                            float frameRate,
                            boolean bigEndian)
```

- Tecnica di codifica dei campioni (SIGNED PCM,UNSIGNED_PCM, A-LAW, U-LAW)
- Frequenza di campionamento (# campioni da catturare per secondo di registrazione)
- Risoluzione audio (# bit da destinare per ciascun campione)
- Numero di canali (modalità mono o stereo)
- Ordine dei byte (BIG ENDIAN, LITTLE ENDIAN)

Relativamente alla codifica di tipo PCM risulta:

`frame:` insieme di byte rappresentanti un singolo campione audio

```
frameSize = (sampleSizeInBits/8) * channels
frameRate = sampleRate
```

Cattura dell'audio da microfono: creazione e apertura della linea TargetDataLine

- Consente la lettura progressiva in tempo reale dei campioni audio provenienti da un dispositivo di input (come un microfono)
- I campioni audio vengono salvati di volta in volta in un buffer di byte interno alla linea
- E' possibile accedere di volta in volta all'array di byte correntemente memorizzati nel buffer
- La dimensione del buffer può essere impostata manualmente o automaticamente

```
DataLine.Info info = new DataLine.Info(TargetDataLine.class,
                                       tmpAf);
 if (!AudioSystem.isLineSupported(info))
    {System.out.println("La linea " + info + " non e'supportata.");}
 else
   try {
      targetDataline = (TargetDataLine) AudioSystem.getLine(info);
      targetDataline.open(tmpAf);
     } catch (LineUnavailableException ex) {
        System.out.println("Impossibile aprire la linea: " + ex);
        return;}
```

Cattura dell'audio da microfono: acquisizione dell'audio in tempo reale dalla linea TargetDataLine

```
 class Capture implements Runnable {
      Thread thread;
      TargetDataLine targetDataline = null;

    public void start() {thread = new Thread(this); thread.start();}
    public void stop() {thread = null;}

  public void run() {
 /* codice per la creazione e apertura della targetDataLine */ […]

ByteArrayOutputStream out = new ByteArrayOutputStream();
int frameSizeInBytes = tmpAf.getFrameSize();
int bufferSubLength = (int)(targetDataline.getBufferSize()/8); //frazione buffer
int bufferLengthInBytes = bufferSubLength * frameSizeInBytes;
byte[] data = new byte[bufferLengthInBytes];
int numBytesRead;
 targetDataline.start();
   while (thread!=null) {
        if((numBytesRead = targetDataline.read(data, 0,
           bufferLengthInBytes)) ==-1) {break;}
             out.write(data, 0, numBytesRead);
           }
  targetDataline.stop();
  targetDataline.close();
}}
```

Riproduzione sonora dell'audio catturato da microfono: creazione di un nuovo oggetto AudioInputStream

Creazione di un nuovo `AudioInputStream` a partire dai campioni audio appena acquisiti.

```
byte audioBytes[] = out.toByteArray();

ByteArrayInputStream bais = new ByteArrayInputStream(audioBytes);
AudioFormat tmpAf = getAudioFormat();
int frameSizeInBytes = tmpAf.getFrameSize();
long lenInFrames = audioBytes.length / frameSizeInBytes;

ais = new AudioInputStream(bais,      // InputStream contenente tutti i campioni
                           tmpAf,    // formato audio dello stream
                           lenInFrames); // numero di frame dello stream
```

dove `out` è l'oggetto **`ByteArrayOutputStream`** impiegato durante la fase di cattura audio

Assegnazione del nuovo `AudioInputStream` alla Clip audio tramite l'invocazione del metodo `setClipFromAudioInputStream(AudioInputStream ais)`

Stefano Leone Monni - *stefano.monni@gmail.com*

Salvataggio della registrazione effettuata: creazione di un file audio

```
public boolean saveAudioFile(AudioInputStream ais,
                             AudioFileFormat.Type fileType,
                             File file)
   {
        if (ais == null) {
            System.out.println("Non esiste alcun segnale audio da salvare");
            return false; }

     try {
           if (AudioSystem.write(ais, fileType, file) == -1) {
               throw new IOException("Problemi nella scrittura del file");}
        } catch (IOException ex) { System.out.println(ex); return false; }

      return true;
    }
```

dove:

- **`AudioInputStream ais =`** audioInputStream dei campioni audio acquisiti
- **`AudioFileFormat.Type fileType =`** tipo di formato di file audio (Es: `FileFormat.Type.WAVE`)
- **`File file =`** file audio su cui andare a scrivere tutti i dati audio

Modifica del segnale audio tramite l'impiego degli oggetti ereditati dalla classe astratta Control

- Ciascuna linea può disporre di uno o più controlli audio che consentono la modifica del segnale sonoro ad essa associato.

- E'possibile interrogare una generica linea per ottenere un elenco di tutti i controlli messi a disposizione

```
public void viewControls(Line myLine)
 {
  Control [] myControls =
              myLine.getControls();

for(int i=0;i<myControls.length;i++)
   System.out.println(i+")Controllo:"
   + myControls[i]);
 }
```

Control	Control.Type	Control.Type instances	
BooleanControl	BooleanControl.Type	**MUTE** mute status of line	**APPLY_REVERB** reverberation on/off
CompoundControl	CompoundControl.Type	(none)	
EnumControl	EnumControl.Type	**REVERB** access to reverb settings (each is an instance of ReverbType)	
FloatControl	FloatControl.Type	**AUX_RETURN** auxiliary return gain on a line **AUX_SEND** auxiliary send gain on a line **BALANCE** left-right volume balance **MASTER_GAIN** overall gain on a line	**PAN** left-right position **REVERB_RETURN** post-reverb gain on a line **REVERB_SEND** pre-reverb gain on a line **SAMPLE_RATE** playback sample rate **VOLUME** volume on a line

Esempio di utilizzo di un oggetto Control: modifica in tempo reale del volume in fase di riproduzione audio


```
public boolean setGain()
  {
  if (mySound !=null) {
  if (!mySound.isControlSupported(FloatControl.Type.MASTER_GAIN))
 {System.out.println("master gain non supportato dalla linea");
  return false;}

 double value = gainSlider.getValue() / 100.0;
 try {
   FloatControl gainControl =
  (FloatControl) mySound.getControl(FloatControl.Type.MASTER_GAIN);
   float dB = (float) (Math.log(value==0.0 ?
               0.0001:value)/Math.log(10.0)*20.0);
   gainControl.setValue(dB);
   } catch (Exception ex) {ex.printStackTrace(); return false;}
 return true;
}
```

dove `mySound` è la linea di tipo `Clip` già vista in precedenza ed impiegata per la fase di riproduzione audio

Esempio di modifica del segnale audio tramite tecniche **DSP** (Digital Signal Processing): Riproduzione del file audio a ritroso (1/2)

```
public boolean reverseWaveForm()
 {
  if (ais==null) return false;
    try{
        ais.reset();
        byte [] samples = new byte [ais.available()];
        ais.read(samples);
        int num_bytes = (int) (ais.getFormat().getSampleSizeInBits() / 8);

        byte [] rev_samples = reverseSamples(samples,num_bytes);

ByteArrayInputStream revStream = new ByteArrayInputStream(rev_samples);
ais = new AudioInputStream(revStream,ais.getFormat(),ais.getFrameLength());
setClipFromAudioInputStream(ais);

   } catch (Exception ex) { ex.printStackTrace(); return false;}
        return true;
 }
```

Stefano Leone Monni - *stefano.monni@gmail.com*

Esempio di modifica del segnale audio tramite tecniche **DSP** (Digital Signal Processing): Riproduzione del file audio a ritroso (2/2)

```
/**
 *Inverte e restituisce l'ordine dei campioni forniti sotto forma di array
 *di byte tenendo conto dei numero di byte rappresentanti ciascun campione
 */
 public byte [] reverseSamples(byte [] samples, int num_bytes)
 {
        if (num_bytes<1) return samples;

        int samples_len = samples.length;
        byte [] reversed = new byte [samples_len];

        int index = 0;

        for(int i=samples_len;i>=num_bytes;i-=num_bytes)
        {
                for (int j=num_bytes-1;j>=0;j--)
                {
                  reversed[index++] = samples[i-j-1];
                }
        }

        return reversed;
 }
```

DSP

Conclusioni

- Il package *javax.sound.sampled* consente una gestione completa dell'audio digitale (dalla fase di cattura audio fino alla sua rielaborazione e riproduzione audio)

- La gestione a basso livello, mediante l'accesso diretto agli stream audio, rende particolarmente agevole la rielaborazione dell'audio mediante tecniche DSP

Java Sound:
Audio digitale dalla teoria alla pratica

Domande?

Bibliografia:

- Java Sound Api Home Page: *http://java.sun.products/java-media/sound/index.html*

- S.L.Monni – *IoProgrammo n.74 (Dicembre 2003) – Un player audio tutto Java*
- S.L.Monni – *IoProgrammo n.75 (Gennaio 2004) – Realizzare un Audio recorder*