UNIVERSITA' DEGLI STUDI DI TRENTO
Facoltà di Scienze Matematiche, Fisiche e Naturali

UNIVERSITY OF TRENTO - Italy

Corso di Laurea Triennale in Informatica

Elaborato Finale

Sviluppo di una libreria per l'acquisizione di video in Java basata su JNI per sostituire QTJava

Relatore:

Prof.
Marco Ronchetti

Laureando:

Matteo Matassoni

Anno accademico 2009 - 2010

A tutte le persone che mi hanno spronato e
a lei che ha sempre creduto in me.

Sommario

Questa tesi verte sullo studio e sviluppo di una libreria software – Java Capture with QuickTime (JCQ) – con funzionalità multimediali per applicativi Java in ambiente Mac OS X. Tale libreria si prefigge lo scopo di sostituire QuickTime for Java (QTJava), almeno per quella parte di funzioni che permettono il processo di cattura video. Proprio per questo, JCQ è composto da un'Application Programming Interface (API)[1] scritta in linguaggio di programmazione Java e che tramite Java Native Interface (JNI) si interfaccia alla libreria QuickTime, sfruttando il *framework* nativo QuickTime Kit Framework (QTKit) per gestire l'acquisizione audio/video da dispositivi digitali, quali videocamere, *iSight* e altre *webcam* e microfoni, spesso integrati nei moderni portatili e computer. I dati catturati vengono digitalizzati e archiviati su disco rigido come file e successivamente possono essere utilizzati e integrati per altri scopi (sharing, post-elaborazione, E-learing, etc.).

L'obiettivo è quello di realizzare un prodotto impiegabile e integrabile con qualsiasi programma Java, che sia di supporto all'intera comunità Java: proprio per questo il codice sorgente della libreria viene rilasciato sotto licenza (da definire), come Open Source.

Una particolare nota d'attenzione è stata posta sulle funzionalità e sulla struttura della libreria: elemento fondamentale affinché risulti completa e semplice per l'utilizzo da parte dei programmatori che intendano farne uso.

Seguirà un'introduzione sul contesto che ha motivato lo sviluppo della libreria; successivamente verranno discusse le tecnologie attualmente esistenti che offrono funzionalità simili e le tecnologie usate durante l'implementazione di JCQ e in supporto ad essa.

[1] insieme di procedure, funzioni, strutture e in alcuni casi oggetti disponibili allo sviluppatore.

Indice

Capitolo 1

Introduzione

Indice

La motivazione che ha portato alla realizzazione della libreria presa in esame nella tesi è la deprecazione di QuickTime for Java (QTJava) (vedi pag. 9). Come verrà spiegato più avanti nella Sezione 2.2, QTJava rappresenta l'unico modo con cui si possa accedere in ambiente Java alle risorse e alle funzioni che l'architettura QuickTime fornisce.

Per colmare questa mancanza, si è deciso di andare a sviluppare una nuova libreria software per Java chiamata Java Capture with QuickTime (JCQ), che grazie all'ausilio di Java Native Interface (JNI), invoca l'Application Programming Interface (API) del QuickTime Kit Framework (QTKit). Tale *framework* è sviluppato da Apple Inc. per il proprio linguaggio di programmazione Objective-C ed è la principale risorsa per l'utilizzo delle funzionalità multimediali di QuickTime in ambiente Cocoa[1] di Mac OS X. A causa di tale limitazione JCQ può funzionare solamente sotto la piattaforma Mac OS X.

[1] ambiente di programmazione orientato agli oggetti sviluppato da Apple Inc. per il sistema operativo Mac OS X.

1.0.1 LODE

Il progetto nasce per esigenze tecniche derivanti dall'utilizzo di QTJava nel software Lectures On DEmand (LODE)[11], sviluppato presso l'Università di Trento da Marco Ronchetti e dal suo gruppo.

> LODE è un sistema sviluppato per l'acquisizione digitale a basso costo di lezioni (specie se basate su presentazioni tipo PowerPoint). L' audio (di buona qualità) ed il video (550x440 pixels, interpolati) possono essere accompagnati da immagini delle slides proiettate in aula (per lezioni basate su Power-Point e simili). Il materiale prodotto può essere distribuito su CD (fino a circa quattro ore di video) oppure su DVD (fino a circa 35 ore di video), o distribuito via web. [...] Lode offre strumenti per navigare interattivamente la lezione (per titolo della slide, altri indici o attraverso una barra temporale). LODE permette anche di attaccare documenti arbitrari a punti generici della lezione.

La prima versione del software LODE fu sviluppata e implementata in Delphi ed era compatibile con Microsoft Windows, ma per motivi di supporto e aggiornamenti successivi si decise di realizzarlo in Java e si provò a renderlo compatibile con il sistema operativo Linux, ma questo non permetteva una gestione ottimizzata dei dispositivi e a livello di codifica si dovevano ricorrere a librerie esterne, complicandone il supporto e la realizzazione. Dopo un'attenta ricerca si decise di passare all'utilizzo di QTJava, che basandosi sull'architettura di QuickTime ne velocizzò l'implementazione grazie alla gestione semplificata di ogni azione necessaria: dalla scelta dei dispositivi disponibili e collegati al computer, alla registrazione e all'esportazione in formato compresso senza complicazioni nella gestione dei *codec*[2]. Essendo QTJava compatibile con i sistemi operativi Mac OS e Microsoft Windows, LODE divenne utilizzabile su entrambe le architetture.

In seguito alla versione 6.1 di QTJava, rilasciata nel 2003, Apple decise di non proseguire più con il suo sviluppo e supporto, cessando perciò di aggiornarne l'API. Con il passare del tempo QTJava divenne sempre di più una libreria datata e

[2]programma o un dispositivo che si occupa di codificare e/o decodificare digitalmente un segnale (tipicamente audio o video) perché possa essere salvato su un supporto di memorizzazione o richiamato per la sua lettura.

superata, in quanto nel frattempo le architetture hardware, i dispositivi digitali e la stessa architettura QuickTime si evolsero. I programmatori che intendono usare QTJava riscontrano problemi affatto semplici, quali messaggi di deprecazione e la compilazione Java sulle moderne architetture a 64 bit, in quanto QTJava è compatibile solamente su quelle a 32 bit.

Lo scopo è realizzare una nuova libreria che possa essere utilizzata da LODE in sostituzione di QTJava utilizzando il QTKit di Apple, un *framework* multimediale basato sì su QuickTime ma nettamente più moderno e al passo con i tempi. Affinché questa libreria divenga un punto di riferimento per tutti coloro che necessitano delle funzionalità da essa fornite e venga continuamente aggiornata e ampliata, si è deciso di rendere disponibile il codice sorgente alla comunità.

Capitolo 2

Tecnologie

Indice

Questo capitolo introduce una prospettiva alle tecnologie analizzate e studiate nella realizzazione di JCQ. In primo luogo, sarà presentata l'architettura di QuickTime, sottolineandone le potenzialità e le funzionalità tecnologiche. In seguito, saranno approfonditi i due principali *framework*: QTJava e QTKit che Apple Inc. mette a disposizione agli sviluppatori per utilizzare QuickTime. La parte finale verterà sulla tecnologia JNI, grazie alla quale è possibile andare ad implementare metodi di classe Java in ambiente nativo, scendendo sì ad un livello di astrazione più basso del Java ma potendone estendere e migliorarne le funzionalità e le performance.

2.1 QuickTime

QuickTime è un'architettura multimediale multipiattaforma per Mac OS e Microsoft Windows, realizzata da Apple Inc., composta da una serie di estensioni multimediali per il sistema operativo, una API consistente e a completa disposizione dei programmatori, un formato digitale di file e un insieme di applicazioni destinate all'utente come il QuickTime Player, il componente ActiveX e il *plug-in* per la riproduzione da browser.

Comunemente a QuickTime viene associata solamente la funzionalità di *media player*[1], ma in realtà è una completa architettura destinata alla multimedialità. Esso, infatti, consente la creazione, la produzione e il supporto di una vasta gamma di formati digitali, tra cui filmati, immagini, suoni, immagini panoramiche, etc.. QuickTime offre il supporto e gestisce l'intero processo dall'inizio alla fine: permette l'acquisizione in tempo reale, la scrittura del flusso di dati, l'importazione e l'esportazione di dati esistenti, l'editing e la composizione, la compressione e in fine la riproduzione.

2.1.1 Un po' di storia

Apple rilasciò la prima versione di QuickTime il 2 dicembre 1991 come componente multimediale aggiuntiva per il sistema operativo System Software versione 6 e successive. La prima dimostrazione pubblica risale al maggio 1991 alla Apple Worldwide Developer Conference (WWDC), in cui Bruce Leak – capo sviluppo del team di Quicktime – riprodusse il famoso spot commerciale – *1984* – del primo Macintosh[1]; tale tecnologia rappresentò una svolta sorprendente per quei tempi.

A prova della flessibilità della tecnologia di Quicktime e dell'avanzata struttura del suo formato digitale, l'ISO approvò, nel 1998, il formato come base per lo standard MPEG-4 [5]. La predisposizione del formato alla codifica, alla trasmissione e all'archiviazione via web erano caratteristiche assenti nel formato MPEG-1, troppo basilare, e all'MPEG-2, che non era stato progettato per la trasmissione via web. La compatibilità con l'MPEG-4 è stata aggiunta con la sesta versione di QuickTime, sviluppata nel 2002. Apple ha tuttavia ritardato per diversi mesi il rilascio del

[1] software per l'esecuzione e la riproduzione di file multimediali.

software per via delle dispute sorte con il comitato sulle licenze del formato digitale. Un compromesso fu poi raggiunto e il software fu rilasciato nel Giugno del 2003.

Importante fu l'aggiornamento alla versione sette di QuickTime che Mac OS v.10.4 introdusse il 29 aprile 2005. Oltre a vari miglioramenti all MPEG-4 e il supporto alla codifica H.264/AVC, venne aggiunto anche il QTKit, un *framework* Cocoa per QuickTime.

L'ultima versione del software è QuickTime X[2], attualmente disponibile unicamente per Mac OS X v.10.6.

2.1.2 I framework

L'API alla base di QuickTime è scritta in C e può quindi essere usata direttamente quando si programma in C o C++, vi sono tuttavia differenze tra la versione di Windows e quella compresa nel *framework* Carbon di Mac OS. Esistono anche interfacce QuickTime per JavaScript e per la piattaforma Windows, quali Visual Basic, C#, e .NET.

Al di sopra dell'API C di QuickTime Apple ha sviluppato anche due *framework* per facilitare l'uso della libreria e renderla accessibile anche a quei linguaggi che seguono l'Object-oriented programming (OOP). Il primo dei due è il QTJava e è rivolto agli sviluppatori Java e sarà discusso nella Sezione 2.2; il secondo, invece, QTKit e è destinato ai programmatori Objective-C/Cocoa per Mac OS X e verrà approfondito nella Sezione 2.3.

[2]la X indica la versione dieci.

Breve Nota

Un framework è una struttura di supporto su cui un software può essere organizzato e progettato. Alla base di esso c'è sempre una serie di librerie di codice utilizzabili con uno o più linguaggi di programmazione, spesso corredate da una serie di strumenti di supporto allo sviluppo del software, come ad esempio un Integrated Development Environment (IDE), un debugger, o altri strumenti ideati per aumentare la velocità di sviluppo del prodotto finito. Lo scopo di un framework è di risparmiare allo sviluppatore la riscrittura di codice già steso in precedenza per compiti simili.

2.2 QuickTime for Java

QTJava, come dice il nome, è una libreria software destinata all'ambiente Java e permette quindi di creare applicazioni con le funzionalità multimediali che l'architettura QuickTime vanta. Questa libreria, in realtà, non è completamente implementata in Java ma rappresenta un *wrapper* Java che, mediante l'utilizzo di JNI, invoca l'API C di QuickTime per crearne e manipolarne le strutture dati. QTJava funziona unicamente per i sistemi operativi Macintosh e Microsoft Windows, in quanto collocata direttamente al di sopra dell'API C di Quicktime, e quindi non è possibile farne uso su altre piattaforme, per esempio Linux.

Apple Inc. ha sfortunatamente smesso di supportare QTJava[2] e l'ultimo aggiornamento alla versione 6.1 fu rilasciato nel 2003; al momento in cui questa analisi è stato compiuta è diventata inutilizzabile: in Microsoft Windows il suo uso solleva l'eccezione "execution protection violation", mentre in Mac OS X l'importazione della libreria genera degli avvisi in quanto la maggior parte dei metodi e delle classi sono state deprecate[3].

La riproduzione di un file con formato compatibile con QuickTime in QTJava, risulta piuttosto semplice: si guardi l'esempio del Listato 2.1. Per prima cosa si sceglie il file da riprodurre, nell'esempio si utilizza un *file dialog*[4] (righe 13-16). Una volta che si è ottenuto il file desiderato per la riproduzione, lo si usa per creare un oggetto `QTFile`, che è la rappresentazione del filmato QuickTime all'interno di QTJava. Una volta aperto il file per la lettura (riga 18-20), si crea la componente grafica nella quale il file verrà riprodotto (riga 22), e la si aggiunge alla finestra (riga 24). Infine, tramite l'istruzione *start()* invocata sul filmato, se ne attiva la riproduzione (riga 26).

Listato 2.1: Riproduzione di un file con formato supportato da QuickTime con QTJava

```
1 import java.io.File;
2 import java.awt.*;
3
4 import quicktime.*;
5 import quicktime.std.movies.Movie;
```

[3] nell'ambito software il termine "deprecazione" significa che le funzioni e i metodi sono stati superati e il loro uso deve essere evitato.

[4] la classica finestra che permette all'utente di scegliere un file dal proprio computer.

```
6  import quicktime.app.view.QTFactory;
7  import quicktime.io.*;
8
9  public class SimpleQTJPlayer extends Frame {
10
11   public SimpleQTJPlayer() throws QTException {
12     // Create a dialog to choose the file to playback
13         FileDialog fd = new FileDialog
14       (this, "SimpleQTJPlayer", FileDialog.LOAD);
15         fd.setVisible(true);
16         File f = new File (fd.getDirectory(), fd.getFile());
17     // Open the file for reading
18         OpenMovieFile omf = OpenMovieFile.asRead(new QTFile(f));
19     // Create the QuickTime movie in memory from the choosen file
20         Movie m = Movie.fromFile(omf);
21     // Create the component from the movie
22         Component c = QTFactory.makeQTComponent(m).asComponent();
23     // Add the component to the frame
24         add(c);
25     // Start the playback of the movie
26         m.start();
27     }
28
29     public static void main (String[] args) {
30         try {
31             QTSession.open();
32             Frame f = new SimpleQTJPlayer();
33             f.pack();
34             f.setVisible (true);
35         } catch (Exception e) {
36             e.printStackTrace();
37         }
38     }
39 }
```

L'acquisizione in QTJava invece viene mostrata nell'esempio del Listato 2.2. Gli oggetti più importanti per l'acquisizione sono il `SequenceGrabber` e il `SGVideoChannel`. Il primo coordina tutti i canali di acquisizione (audio, video, etc.), e permette di impostare alcuni parametri come il salvataggio dei dati acquisiti su file, il tempo massimo di acquisizione, etc. Il secondo, invece assiste solamente nella parte del canale video.

Per un'ulteriore approfondimento su QTJava, si rimanda alla sua documentazione Javadoc[8].

Listato 2.2: Acquisizione con QTJava

```
1  import java.io.File;
2
3  import quicktime.*;
4  import quicktime.io.*;
5  import quicktime.std.*;
6  import quicktime.std.sg.*;
7  import quicktime.std.movies.*;
8  import quicktime.std.image.*;
9  import quicktime.qd.*;
10 import quicktime.sound.*;
11 import java.awt.*;
12 import java.awt.event.*;
13 import javax.swing.Timer;
14
15 public class VideoCapture extends Frame
16   implements ActionListener {
17   SequenceGrabber grabber;
18   SGVideoChannel videoChannel;
19   QDGraphics gw;
20   QDRect grabBounds;
21   boolean recording;
22   Button stopButton;
23   QTFile grabFile;
24   public VideoCapture() throws QTException {
25       super("Video Capture");
26       stopButton = new Button("Stop");
27       stopButton.addActionListener(this);
28       add(stopButton);
29       initRecording();
30   }
31   public void actionPerformed(ActionEvent e) {
32       if(e.getSource() == stopButton) {
33           try {
34               recording = false;
35               if(grabber != null) {
36                   grabber.stop();
37                   grabber.release();
38               }
39           } catch(Exception ex) {
40               ex.printStackTrace();
41           } finally {
42               // quit the app
43               System.exit(0);
44           }
45       }
46   }
47
48   protected void initRecording() throws QTException {
```

```
49        grabber = new SequenceGrabber();
50        // force an offscreen gworld
51        grabBounds = new QDRect(320, 240);
52        gw = new QDGraphics(grabBounds);
53        grabber.setGWorld(gw, null);
54        // get videoChannel and set its bounds
55        videoChannel = new SGVideoChannel(grabber);
56        videoChannel.setBounds(grabBounds);
57        // get settings
58        videoChannel.settingsDialog();
59        videoChannel.setUsage(StdQTConstants.seqGrabRecord);
60        // prepare for record only
61        grabber.prepare(false, true);
62        grabber.startPreview();
63        // create output file
64        grabFile = new QTFile(new File("MyMovie.mov"));
65        // data are writed on the disk
66        grabber.setDataOutput(grabFile, StdQTConstants.seqGrabToDisk);
67        grabber.startRecord();
68        recording = true;
69        // set up thread to idle
70        ActionListener timerCallback =
71            new ActionListener() {
72                public void actionPerformed(ActionEvent e) {
73                    if(recording) {
74                        try {
75                            grabber.idle();
76                            grabber.update(null);
77                        } catch(QTException qte) {
78                            qte.printStackTrace();
79                        }
80                    }
81                }
82            };
83        Timer timer = new Timer(50, timerCallback);
84        timer.start();
85    }
86    public static void main(String[] args) {
87        try {
88          QTSession.open();
89            Frame f = new VideoCapture();
90            f.pack();
91            f.setVisible(true);
92        } catch(QTException qte) {
93            qte.printStackTrace();
94        }
95    }
96 }
```

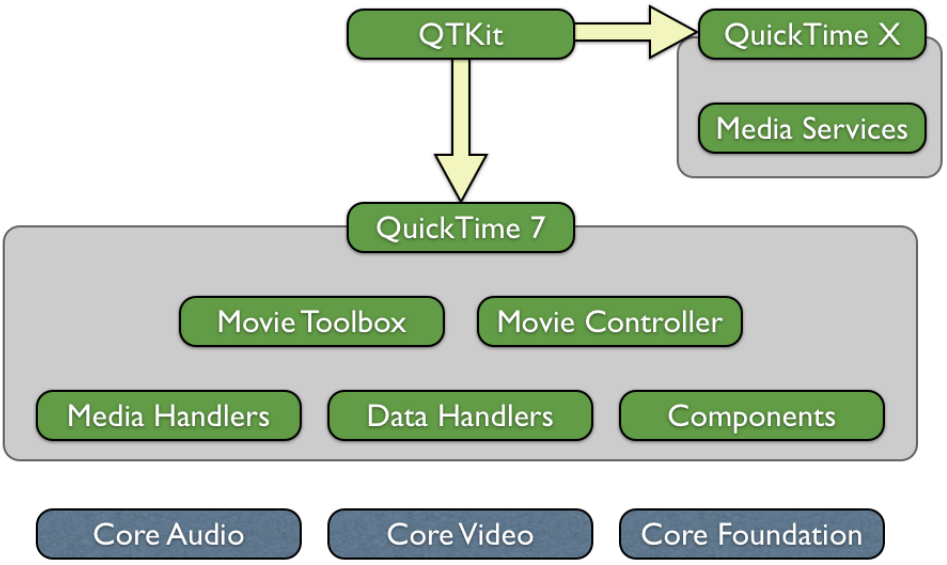

Figura 2.1: Il QTKit è sviluppato al di sopra di QuickTime.

2.3 QuickTime Kit Framework

QTKit[6] (`QTKit.framework`) è un *framework* Objective-C e fornisce ai programmatori Cocoa l'accesso facilitato alle funzionalità dell'architettura QuickTime. Essendo implementato in linguaggio di programmazione obj-c[5] e costruito direttamente al di sopra della libreria C di QuickTime, come mostrato in Figura 2.1, non è necessario per gli sviluppatori conoscere i meccanismi e i funzionamenti a basso livello.

Ogni *major release* di Mac OS X è accompagnata da notevoli miglioramenti e da un'ampia estensione del QTKit, in cui ne viene arricchita l'API, introducendo nuovi metodi e funzioni e con l'aggiunta del supporto a nuovi formati e *codec*.

2.3.1 Architettura del framework

Il *framework* QTKit è composto da ben 23 classi e più di 400 tra metodi, attributi, costanti, e notifiche Objective-C/Cocoa. Per rendere maggiormente comprensibile

[5]abbreviazione di Objective-C.

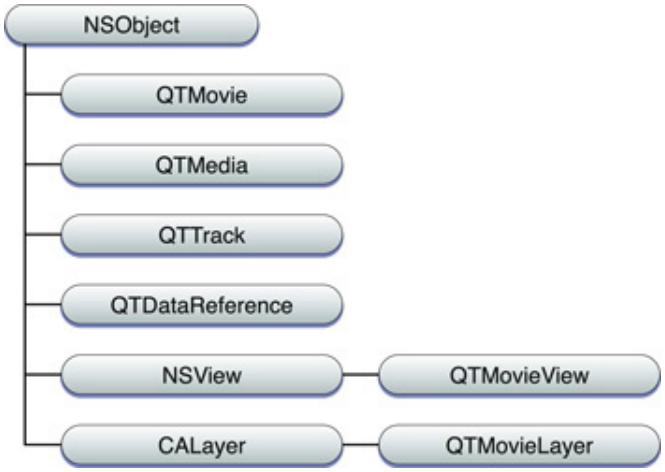

Figura 2.2: Panoramica sulla gerarchia di alcune classi di QTKit per la visualizzazione, riproduzione e modifica.

la struttura interna dell'API del *framework*, la sua gerarchia può essere suddivisa in due gruppi principali:

- il primo composto dalle sei classi mostrate in Figura 2.2, che si occupano principalmente della visualizzazione, della riproduzione e della modifica di filmati QuickTime.

- il secondo composto dai protocolli[6] e dalle classi in Figura 2.3 che permettono l'acquisizione e la cattura da dispositivi multimediali.

La riproduzione in QTKit

In primis verranno approfonditi i principali oggetti che l'API di QTKit rende disponibili in supporto al programmatore Cocoa per effettuare la riproduzione dei file all'interno di applicativi per Mac OS X.

QTMovie È l'oggetto principale per questo *smallset*[7] del *framework* e rappresenta il filmato di QuickTime.

[6] il *protocol* in obj-c corrisponde alla *interface* di Java.
[7] porzione, parte.

QTMovieView Eredita dalla `NSView` e quindi si comporta come tutte le *view* di Cocoa. Viene tipicamente usata in combinazione con l'oggetto `QTMovie` per visualizzare, gestire, riprodurre e modificare il filmato.

QTTrack Un `QTMovie` può averne più di una e rappresenta la singola traccia, come quella audio o video. È possibile leggere e modificare le proprietà che essa incapsula.

QTMedia È associato all'oggetto `QTTrack` e definisce il tipo di quest'ultima. È possibile leggere e modificare le proprietà che essa incapsula, ma solo quando si usa QuickTime 7.

QTDataReference Definisce dove si trova l'intero filmato Quicktime o parte dei sui dati.

QTMovieLayer Eredita da CALayer e fornisce un *layer*[8] grafico del Core Animation *framework* sulla quale l'oggetto `QTMovie` può venire renderizzato.

La riproduzione di un file con formato supportato da QuickTime in Cocoa tramite QTKit risulta molto semplice, in quanto basta combinare la componente grafica `QTMovieView` e l'oggetto che rappresenta il filmato QuickTime aperto `QTMovie`, come mostrato nell'esempio del Listato 2.3.

Listato 2.3: Riproduzione di un file con formato supportato da QuickTime con QTKit

```
1
2 #import <Cocoa/Cocoa.h>
3 #import <QTKit/QTKit.h>
4
5 @interface QTKitSimplePlayerAppDelegate : NSObject <NSApplicationDelegate> {
6     NSWindow *window;
7   QTMovieView *movieView;
8   QTMovie *movie;
9 }
10
11 @property (assign) IBOutlet NSWindow *window;
12 @property (retain) IBOutlet QTMovieView *movieView;
13
```

[8] nella grafica bidimensionale rappresenta una porzione di schermo, sulla quale è possibile disegnare oggetti.

```
14 @end
15
16
17 @implementation QTKitSimplePlayerAppDelegate
18
19 @synthesize window, movieView;
20
21 - (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
22   // Create the File Open Dialog class.
23   NSOpenPanel* openDlg = [NSOpenPanel openPanel];
24
25   // Enable the selection of files in the dialog.
26   [openDlg setCanChooseFiles:YES];
27
28   // Display the dialog.  If the OK button was pressed,
29   // process the files.
30   if ([openDlg runModalForDirectory:nil file:nil] == NSOKButton) {
31     // Get choosen file's URL
32     NSURL* fileURL = [[openDlg URLs] objectAtIndex:0];
33
34     NSError *error = nil;
35     movie = [[QTMovie alloc] initWithURL:fileURL error:&error];
36     if (movie != nil) {
37       [movieView setMovie:movie];
38       [movie autoplay];
39     }
40     else {
41       NSLog(@"Error: %@", [error localizedDescription]);
42     }
43   }
44 }
45
46 - (void)dealloc {
47   [super dealloc];
48   [movie release];
49   [movieView release];
50 }
51
52 @end
```

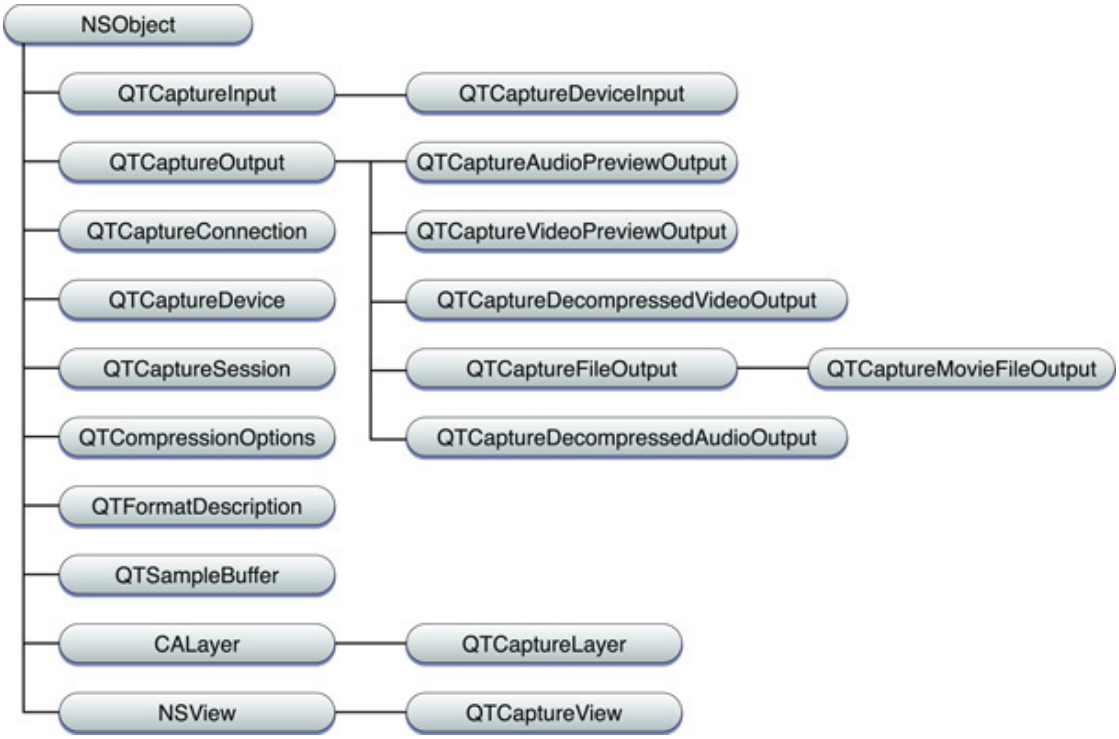

Figura 2.3: Gerarchia delle classi per la cattura di QTKit.

L'acquisizione in QTKit

Le API del QTKit per l'acquisizione sono ottimizzate per fornire una cattura in tempo reale da dispositivi, quali *iSight*[9], videocamere, microfoni, magnetofoni[10] e interfacce audio professionali e non. I dati acquisiti possono essere salvati come filmati compressi, visualizzati sotto forma di anteprima e possono essere esportati in formato *raw*[11] per una successiva rielaborazione.

Analizziamo nel dettaglio tutte le strutture e gli oggetti che le API di QTKit mettono a disposizione.

QTCaptureSession Componente primario per la cattura del flusso dei dati; controlla lo stato (cattura attiva o non) e gestisce le connessioni tra gli input

[9] webcam prodotta da Apple Inc., incorporata nei recenti modelli di macbook, macbook pro e iMac o disponibile come dispositivo esterno per gli altri Macintosh.

[10] registratori magnetici.

[11] letteralmente "grezzi", in questo contesto significa che i fotogrammi catturati non vengono compressi e non c'è alcuna perdita nel processo di digitalizzazione.

e gli output che le sono stati associati.

QTCaptureInput È un'astrazione, cioè un protocollo[12] nell'ambiente Cocoa, che rappresenta una sorgente in entrata per la cattura. Si utilizzano classi specifiche che implementano effettivamente i metodi di questo protocollo, come la `QTCaptureDeviceInput`, che rappresenta un input dato da dispositivo multimediale.

QTCaptureOutput Altro protocollo che definisce tutti i meccanismi che una sorgente in uscita fornisce. Si utilizzano classi specifiche che implementano questo protocollo, come la `QTCaptureMovieFileOutput`, che scrive il flusso di dati su file con il formato e la compressione impostata tramite (`QTCompressionOptions`).

QTCaptureConnection Identifica una connessione nella quale i dati vengono spedito dall'input alla sessione, e a sua volta dalla sessione all'output.

QTCaptureView Componente grafico (view) che permette la visione dell'anteprima video che la `QTCaptureSession` sta processando.

Listato 2.4: Acquisizione e anteprima con QTKit

```
1
2  #import <Cocoa/Cocoa.h>
3  #import <QTKit/QTkit.h>
4
5  @interface MyRecorderController : NSObject {
6
7      IBOutlet QTCaptureView *mCaptureView;
8
9      QTCaptureSession *mCaptureSession;
10     QTCaptureMovieFileOutput *mCaptureMovieFileOutput;
11     QTCaptureDeviceInput *mCaptureVideoDeviceInput;
12     QTCaptureDeviceInput *mCaptureAudioDeviceInput;
13 }
14
15 - (IBAction)startRecording:(id)sender;
16 - (IBAction)stopRecording:(id)sender;
17
18 @end
19
```

[12] per chi conosce il Java è quello che viene indicato con `interface`.

```
@interface MyRecorderController()
- (void)handle:(NSError *)error;
@end


@implementation MyRecorderController

- (void)awakeFromNib {
  // Create the capture session
  mCaptureSession = [[QTCaptureSession alloc] init];

  BOOL success = NO;
  NSError *error;

  // Find a video device
    QTCaptureDevice *videoDevice = [QTCaptureDevice
                  defaultInputDeviceWithMediaType:QTMediaTypeVideo];
    success = [videoDevice open:&error];

  // If a video input device can't be found or opened, try to find and open a muxed
          input device
  if (!success) {
    videoDevice = [QTCaptureDevice
             defaultInputDeviceWithMediaType:QTMediaTypeMuxed];
    success = [videoDevice open:&error];
    }

    if (!success) {
        videoDevice = nil;
        // Handle error
    [self handle:error];
    }

    if (videoDevice) {
    //Add the video device to the session as a device input
    mCaptureVideoDeviceInput = [[QTCaptureDeviceInput alloc]
                  initWithDevice:videoDevice];
    success = [mCaptureSession addInput:mCaptureVideoDeviceInput
                    error:&error];
    if (!success) {
      // Handle error
      [self handle:error];
    }

    // If the video device doesn't also supply audio, add an audio device input to
            the session
        if (![videoDevice hasMediaType:QTMediaTypeSound] &&
      ![videoDevice hasMediaType:QTMediaTypeMuxed]) {
            QTCaptureDevice *audioDevice = [QTCaptureDevice
                      defaultInputDeviceWithMediaType:QTMediaTypeSound];
```

```
68            success = [audioDevice open:&error];
69
70            if (!success) {
71                audioDevice = nil;
72                // Handle error
73        [self handle:error];
74            }
75
76            if (audioDevice) {
77                mCaptureAudioDeviceInput = [[QTCaptureDeviceInput alloc]
78                      initWithDevice:audioDevice];
79                success = [mCaptureSession addInput:mCaptureAudioDeviceInput
80                        error:&error];
81                if (!success) {
82                    // Handle error
83                }
84            }
85        }
86
87    // Create the movie file output and add it to the session
88        mCaptureMovieFileOutput = [[QTCaptureMovieFileOutput alloc] init];
89        success = [mCaptureSession addOutput:mCaptureMovieFileOutput error:&error];
90        if (!success) {
91            // Handle error
92      [self handle:error];
93        }
94
95  // Set the compression for the audio/video that is recorded to the hard disk.
96  NSEnumerator *connectionEnumerator = [[mCaptureMovieFileOutput connections]
          objectEnumerator];
97  QTCaptureConnection *connection;
98
99  // iterate over each output connection for the capture session and specify the
          desired compression
100 while ((connection = [connectionEnumerator nextObject])) {
101   NSString *mediaType = [connection mediaType];
102   QTCompressionOptions *compressionOptions = nil;
103   if ([mediaType isEqualToString:QTMediaTypeVideo]) {
104     // use H.264
105     compressionOptions = [QTCompressionOptions
106                 compressionOptionsWithIdentifier:@"
                            QTCompressionOptions240SizeH264Video"];
107   } else if ([mediaType isEqualToString:QTMediaTypeSound]) {
108     // use AAC Audio
109     compressionOptions = [QTCompressionOptions
110                 compressionOptionsWithIdentifier:@"
                            QTCompressionOptionsHighQualityAACAudio"];
111   }
112
113   // set the compression options for the movie file output
```

```
114     [mCaptureMovieFileOutput setCompressionOptions:compressionOptions
115                         forConnection:connection];
116   }
117     // Associate the capture view in the UI with the session
118         [mCaptureView setCaptureSession:mCaptureSession];
119
120     // Start session
121         [mCaptureSession startRunning];
122   }
123
124 }
125
126 - (void)windowWillClose:(NSNotification *)notification {
127   // stop session
128   [mCaptureSession stopRunning];
129
130   // close the devices
131     if ([[mCaptureVideoDeviceInput device] isOpen])
132         [[mCaptureVideoDeviceInput device] close];
133     if ([[mCaptureAudioDeviceInput device] isOpen])
134         [[mCaptureAudioDeviceInput device] close];
135
136 }
137
138 - (void)dealloc {
139   [mCaptureSession release];
140   [mCaptureVideoDeviceInput release];
141     [mCaptureAudioDeviceInput release];
142   [mCaptureMovieFileOutput release];
143
144   [super dealloc];
145 }
146
147 // Action from the UI
148 - (IBAction)startRecording:(id)sender {
149   [mCaptureMovieFileOutput recordToOutputFileURL:
150    [NSURL fileURLWithPath:@"/Users/Shared/My Recorded Movie.mov"]];
151 }
152
153 - (IBAction)stopRecording:(id)sender {
154   [mCaptureMovieFileOutput recordToOutputFileURL:nil];
155 }
156
157 // Helper method to handle the error
158 - (void)handle:(NSError *)error {
159   NSLog(@"Error: %@", [error localizedDescription]);
160 }
161
162 @end
```

2.4 Java e la Java Native Interface

In generale, la piattaforma di programmazione Java fornisce allo sviluppatore tutto quello di cui ha bisogno nei vari settori, e in molti di questi (applicativi web, applicativi desktop, programmazione distribuita, etc.) permette di ottenere ottimi risultati in un tempo significativamente inferiore ad altri linguaggi.

Tuttavia, esistono casi in cui è indispensabile che un'applicazione Java interagisca a basso livello con il sistema in cui viene eseguita, cioè con la sua parte nativa. Per questo l'ambiente di sviluppo di Java contiene al suo interno la libreria JNI. Si tratta di una collezione di librerie che consentono a programmi Java di far uso di servizi implementati tramite codice nativo del sistema operativo nella quale la Java Virtual Machine (JVM) risiede. Tipicamente, l'interazione con la parte a basso livello del sistema operativo è realizzata comunicando con altri linguaggi, tra i quali il C, C++, obj-c e assembly.

L'interazione a basso livello con il sistema non è sempre consigliata – soprattuto nello sviluppo di nuove applicazioni – e in un certo senso può sembrare in contrasto con l'etica di Java, ma in alcuni casi può rendersi estremamente utile o addirittura indispensabile.

Fra i più importanti si possono menzionare:

- **Riusabilità**. Capita che nuove applicazioni Java debbano interfacciarsi con software scritto in linguaggio di programmazione diverso da Java e non più convertibile per le più disparate ragioni, quali fattibilità, costi o vincoli di carattere tecnico.

- **Estensibilità**. Aggiungere ai programmi funzionalità che non sono messe a disposizione dall'ambiente di programmazione Java.

- **Performance**. Alcune delle funzioni che Java offre non sono sempre ottimizzate a livello di calcolo computazione e potrebbe presentarsi la necessità di disporre di particolari performance.

Se da un lato si ottengono molteplici vantaggi dall'uso di JNI, dall'altro però può venire compromessa la portabilità che il linguaggio Java vanta. L'unico modo per assicurare la piena portabilità di questi applicativi è quello di implementare la

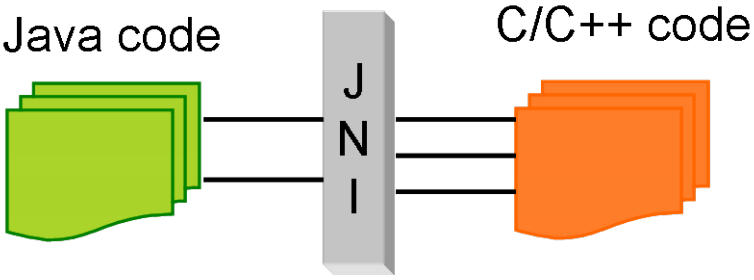

Figura 2.4: JNI permette lo scambio fra Java e il codice nativo.

parte di software – in particolare la libreria – che interagisce a basso livello per tutti i sistemi operativi a cui si intende offrirne supporto. In questo modo, anche se non si può affermare di aver scritto un'applicazione puramente in Java, se ne potranno comunque sfruttare tutti i vantaggi, mantenendone la portabilità. Può sembrare una contraddizione, ma d'altra parte Java stesso è realizzato così. Esistono, infatti, molteplici JVM, ognuna diversamente implementata per il sistema operativo a cui è destinata.

Conseguenza principale dell'uso di JNI è quella di ottenere un prodotto che non può essere considerato cento per cento Java.

2.4.1 Struttura e funzionalità di JNI

Come si è visto nel paragrafo precedente, JNI è il tramite tra i due "mondi", quello di una JVM e quello nativo a basso livello del sistema operativo in cui essa "vive". In Figura 2.4 è messo in risalto proprio questo aspetto.

I tipi di dato in JNI

Come già detto in precedenza, è possibile possibile chiamare e interagire con la parte a basso livello andando ad implementare in codice nativo i metodi delle classi Java. In realtà, è possibile anche fare l'esatto contrario: creare una JVM e invocarne i metodi di classi di sistema o anche definite dall'utente. Il Java e il C/C++ sono linguaggi che, come si sa, condividono in parte la stessa sintassi. Bisogna, però, tenere conto che malgrado i tipi primitivi abbiano la stessa nomenclatura, non sempre il valore associato viene interpretato nello stesso modo. Ad esempio, in

Tipi Java	Tipi nativi	Descrizione
`boolean`	`jboolean`	unsigned 8 bits
`byte`	`jbyte`	signed 8 bits
`char`	`jchar`	unsigned 16 bits
`short`	`jshort`	signed 16 bit
`int`	`jint`	signed 32 bits
`long`	`jlong`	signed 64 bits
`float`	`jfloat`	32 bits
`double`	`jdouble`	64 bits
`void`	`jvoid`	N/A

Tabella 2.1: Mapping tra tipi primitivi di Java e il codice nativo.

Java non esistono gli interi `unsigned`[13] ma soltanto quelli con segno, quindi un numero intero a 32 bit in C/C++ senza segno maggiore di 2147483647, se passato ad un programma Java, deve essere contenuto in un intero lungo a 64 bit. Per mantenere la bidirezionalità tra i due ambienti, nella Java Native Interface sono definiti dei tipi standard. La Tabella 2.1 riporta i tipi di dato primitivi di Java e i corrispondenti j<Type>, e in Figura 2.5 vengono riportati i tipi referenziati (array, classi, stringhe, super oggetto, etc.).

Per comprendere meglio il *mapping* tra i tipi si guardi l'esempio del seguente metodo Java:

```
private native String getNativeLocalizedDisplayName();
```

che ritorna una `java.lang.String` e nella parte nativa:

```
JNIEXPORT jstring JNICALL
Java_it_unitn_science_JCQLibrary_CaptureDevice_
getNativeLocalizedDisplayName
(JNIEnv *env, jobject caller) {}
```

ritorna una `jstring`.

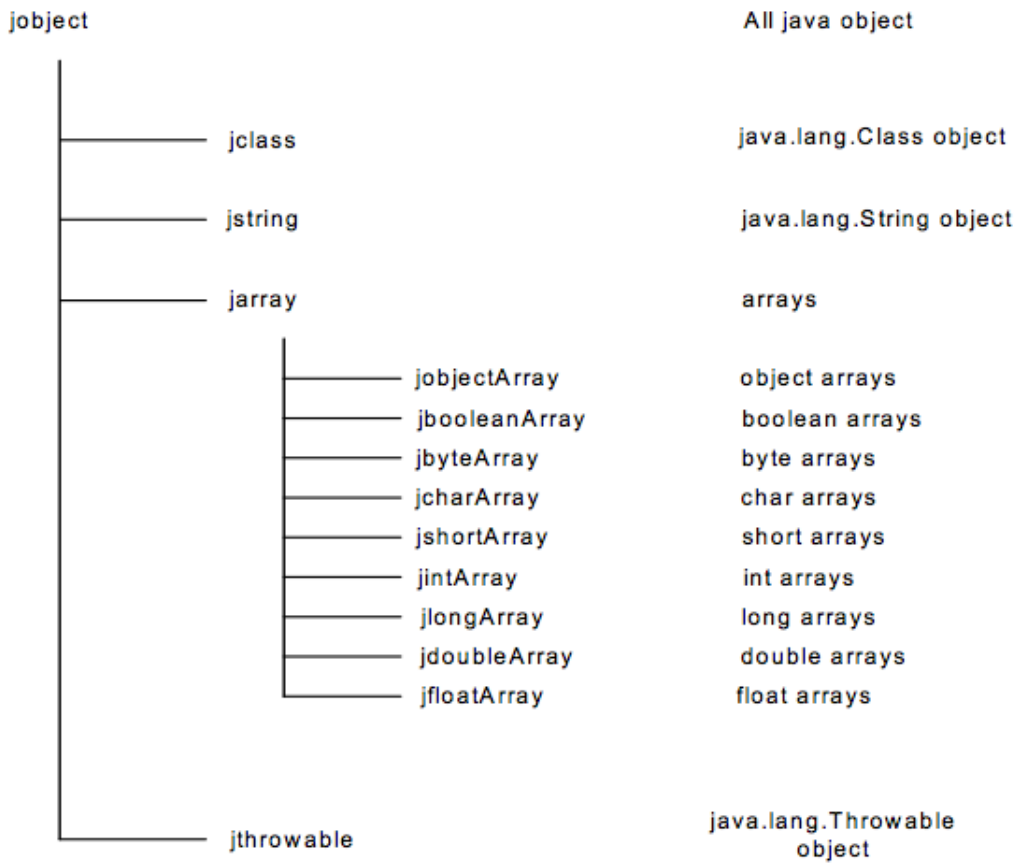

Figura 2.5: Mapping tra la gerarchia dei tipi referenziati di Java e il codice nativo.

Accesso agli attributi e chiamata dei metodi

Un'altra cosa di cui tenere conto per quanto riguarda la JNI sono le cosiddette *type signatures*[14] , che sono riportate in Tabella 2.2, indicano con un codice convenzionale il tipo di parametri e il tipo di ritorno di un qualunque metodo Java e vengono utilizzate dalle istruzioni native per attribuire il tipo corretto della controparte Java.

Ad esempio il metodo:

```
long myMethod(int n, String s, int[] arr)
```

[13] senza segno.

[14] letteralmente firme del tipo di dato.

Type Signature	Java Type
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
V	void
L<fully-qualified-class>;	fully-qualified-class
[type	type[]
(arg-types) ret-type	method type

Tabella 2.2: "Firme" dei tipi nella Java VM

avrà come *signature*:

```
(ILjava/lang/String;[I)J
```

Breve Nota

È possibile utilizzare il disassemblatore javap per ottenere le firme dei metodi di una classe.
Per esempio: `javap -s java.lang.String`.

Le "firme" diventano importanti quando si vuole accedere agli attributi o invocare un metodo Java dall'ambiente nativo. Nel caso in cui si desidera accedere al valore di un attributo per prima cosa bisogna recuperarne il *field ID* invocando, a seconda che l'attributo sia statico o meno, la funzione `Get<Type>Field(...)` o quella `GetStatic<Type>Field(...)`, che come si vedrà negli esempi successivi richiedono oltre, l'environment, la classe, il nome e la firma del tipo dell'attributo. Una volta che si è ottenuto l'identificatore dell'attributo si utilizzano le funzioni `Get<Type>Field(...)` o `GetStatic<Type>Field(...)` a cui si passa l'environment, l'istanza o la classe se si è nel caso statico e il *field ID*, per ottenerne il valore.

Nello stesso modo funzionano le chiamate ai metodi: prima si ottiene il *method ID* tramite nome del metodo e la firma del suo tipo, e successivamente si invoca il metodo tramite la funzione `Call<ret-type>Method(...)`, nel caso di chiamata ad un metodo di un'istanza, o `CallStatic<ret-type>Method(...)`, nel caso di metodo statico di una classe .

Esempi

L'esempio del Listato 2.5 e Listato 2.6 mostra come effettuare l'accesso ad un attributo di una classe Java dalla parte nativa tramite JNI.
Per compilare il tutto bisogna:

- compilare la parte nativa

```
cc -c -I/System/Library/Frameworks/JavaVM.framework/Headers
StaticFieldAccess_Native.c
```

- creare e assemblare la parte nativa compilata per formare libreria JNI

```
cc -dynamiclib -o libStaticFieldAccess.jnilib StaticFieldAccess_Native.o
-framework JavaVM
```

- compilare la parte java

```
javac StaticFieldAccess.java
```

- eseguire il programma Java impostando la library path dove si trova la jnilib

```
java -Djava.library.path=`pwd` StaticFieldAccess
```

Listato 2.5: Esempio di accesso ad un attributo Java dalla parte nativa: definizione della classe Java

```
1  class StaticFieldAccess {
2    static {
3      System.loadLibrary("StaticFieldAccess");
4    }
5
6      private static int si;
7
8      private native void accessField();
9      public static void main(String args[]) {
10     StaticFieldAccess c = new StaticFieldAccess();
11     StaticFieldAccess.si = 100;
12     c.accessField();
13     System.out.println("In Java:");
14     System.out.println("  StaticFieldAccess.si = " + si);
15      }
16   }
```

Listato 2.6: Esempio di accesso ad un attributo Java dalla parte nativa: implementazione nativa in C

```
1  #include <jni.h>
2  #include <stdio.h>
3
4  JNIEXPORT void JNICALL
5   Java_StaticFieldAccess_accessField(JNIEnv *env, jobject obj)
6   {
7       jfieldID fid;   /* store the field ID */
8       jint si;
9
10      /* Get a reference to obj's class */
11      jclass cls = (*env)->GetObjectClass(env, obj);
12
13      printf("In C:\n");
14
15      /* Look for the static field si in cls */
16      fid = (*env)->GetStaticFieldID(env, cls, "si", "I");
17      if (fid == NULL) {
18          return; /* field not found */
19      }
20      /* Access the static field si */
21      si = (*env)->GetStaticIntField(env, cls, fid);
22      printf("  StaticFieldAccess.si = %d\n", si);
23      (*env)->SetStaticIntField(env, cls, fid, 200);
24   }
```

Si può facilmente comprendere che il programma avrà come output:

```
In C:
    StaticFieldAccess.si = 100
In Java:
    StaticFieldAccess.si = 200
```

Mentre, l'esempio del Listato 2.7 e Listato 2.8 mostra come chiamare metodi di una classe Java dalla parte nativa tramite JNI.
Per compilare il tutto bisogna:

- compilare la parte nativa

  ```
  cc -c -I/System/Library/Frameworks/JavaVM.framework/Headers
  InstanceMethodCall_Native.c
  ```

- creare e assemblare la parte nativa compilata per formare libreria JNI

  ```
  cc -dynamiclib -o libInstanceMethodCall.jnilib InstanceMethodCall_Native.o
  -framework JavaVM
  ```

- compilare la parte java

  ```
  javac InstanceMethodCall.java
  ```

- eseguire il programma Java impostando la library path dove si trova la jnilib

  ```
  java -Djava.library.path=`pwd` InstanceMethodCall
  ```

Listato 2.7: Esempio di chiamata di un metodo Java dalla parte nativa: definizione della classe Java

```
1  class InstanceMethodCall {
2      private native void nativeMethod();
3      private void callback() {
4          System.out.println("In Java");
5      }
6      public static void main(String args[]) {
7          InstanceMethodCall c = new InstanceMethodCall();
8          c.nativeMethod();
9      }
10     static {
11         System.loadLibrary("InstanceMethodCall");
12     }
```

```
13 }
```

Listato 2.8: Esempio di chiamata di un metodo Java dalla parte nativa: implementazione nativa in C

```
1  #include <jni.h>
2  #include <stdio.h>
3
4  JNIEXPORT void JNICALL
5   Java_InstanceMethodCall_nativeMethod(JNIEnv *env, jobject obj)
6   {
7       jclass cls = (*env)->GetObjectClass(env, obj);
8       jmethodID mid =
9           (*env)->GetMethodID(env, cls, "callback", "()V");
10      if (mid == NULL) {
11          return; /* method not found */
12      }
13      printf("In C\n");
14      (*env)->CallVoidMethod(env, obj, mid);
15  }
```

Che avrà come output:

```
In C

In Java
```

Capitolo 3

Implementazione

Indice

Questo capitolo tratta del soggetto principale della tesi: la libreria JCQ; verranno descritte le principali tecniche con la quale è stata realizzata, quali le scelte di implementazione legate alla gestione della memoria e l'utilizzo di componenti grafiche di Cocoa per la realizzazione di nuove componenti Abstract Window Toolkit (AWT)/Swing di Java.

Infine, verrà trattata una parte in cui si discuterà della compatibilità tra JCQ e QTJava.

3.1 Gestione della memoria

3.1.1 Gestione della memoria in Objective-C

Nell'ambiente Objective-C/Cocoa, gli oggetti sono creati continuamente e messi a disposizione di altri oggetti; per questo, bisogna essere sempre sicuri di rilasciarli (deallocarli) correttamente, in modo da non fare occupare più spazio di memoria (*memory leaks*) del dovuto dalla propria applicazione.

Apple ha introdotto il sistema di gestione chiamato **Retain Counting**, basato sul *reference counting*[1]: un oggetto esiste finché il suo *retain count* non diventa 0; se questo accade, l'oggetto è automaticamente deallocato, cioè lo spazio occupato in memoria viene rilasciato, perché non è più utile.

Un oggetto viene creato con il metodo `alloc` e si distrugge con il metodo `dealloc`. Vediamo un esempio per chiarire:

```
1 MyClass *anInstance = [[MyClass alloc] init];
2 [anInstance retain];
3 [anInstance release];
4 [anInstance release];
```

I metodi `retain` e `release` rispettivamente incrementano e decrementano il *retain count* dell'oggetto a cui vengono inviati, in questo caso `anInstance`. Il metodo `alloc` istanzia l'oggetto e associa al suo *retain count* il valore 1. Per cui, dal valore iniziale di 1, il *retain count* passa a 2 (seconda riga) a 1 (terza riga) e infine a 0 (quarta riga). Di conseguenza l'oggetto `anInstance` viene distrutto e rilasciato subito dopo la `release` che manda a 0 il *retain count*, chiamando automaticamente il metodo `dealloc` dell'oggetto.

[1] contatore dei riferimenti: tecnica di memorizzazione del numero di riferimenti, puntatori o handle a una risorsa, come un oggetto o un blocco di memoria.

Breve Nota

Il metodo ***dealloc*** *non deve mai essere invocato dal programmatore, quindi non si scriverà mai* `[anInstance dealloc]`.

Autorelease

In Cocoa esiste anche il metodo `autorelease`. In altre parole, quando un oggetto è autorilasciato, significa che l'*autorelease pool* gli invierà a breve un `release`. È come se si stesse prolungando la vita di un oggetto, in modo da poterne approfittare utilizzandolo per qualche scopo, per esempio l'`autorelease` è utilizzato nei metodi che devono ritornare degli oggetti e sarà il chiamante, eventualmente, a farne una `retain`. Più precisamente, in Cocoa, un *autorelease pool* è creato automaticamente dall'applicazione prima di gestire un particolare evento ed il suo rilascio avviene al termine dell'esecuzione dello stesso. Ad esempio, eventi di tracking o click del mouse, etc.. Quindi, non bisogna preoccuparsi della gestione dei pool, se non in particolari casi, come cicli for o while in cui si allocano molti oggetti temporanei.

Breve Nota

Objective-C 2.0 ha introdotto anche la ***garbage collection****, ma in maniera dipendente dal sistema operativo.*

3.1.2 Gestione della memoria in Java

In Java gli oggetti vengono allocati dinamicamente tramite la chiamata del costruttore di classe preceduta dall'operatore `new`, per esempio:

```
Classname anInstance = new Classname();
```

Contrariamente a quanto si è visto per l'obj-c, in Java è il **Garbage Collector** a gestire il rilascio della memoria dell'oggetto non più referenziato. Inoltre ogni classe Java può implementare al proprio interno il metodo `finalize`, privo di argomenti, e un istante prima che l'oggetto venga eliminato dal Garbage Collector, la JVM lo richiamerà per eliminare qualsiasi risorsa rimanente.

3.1.3 Gestione della memoria in JCQ

La libreria JCQ che è stata sviluppata, come è già stato detto, è un *wrapper* Java costruito attorno all'API nativa del *framework* Cocoa QTKit. Bisogna tenere conto, quindi, che operando sia in Java che a livello nativo con Cocoa, la gestione della memoria deve essere seguire i paradigmi di entrambi i linguaggi, come descritto nei due paragrafi precedenti.

Per questo motivo nelle classi Java della libreria che si appoggiano direttamente agli oggetti Cocoa di QTKit il costruttore di classe invoca una *subroutine* nativa nella quale si alloca l'oggetto nativo e se ne ritorna il puntatore (riga 7) in modo da poterne salvare la referenza in un campo all'interno dell'istanza Java che si sta allocando. Successivamente la referenze può essere recuperata quando si dovranno chiamare i metodi dell'oggetto nativo. Nello stesso modo si è implementato il metodo Java `finalize`: si chiama una *subroutine* nativa che recupera l'oggetto Cocoa dalla referenza salvata nella classe Java e ne rilascia lo spazio di memoria invocando una `release` (riga 11), così l'oggetto nativo viene distrutto quando il corrispettivo in Java non è più utilizzato e viene liberato dal Garbage Collector, evitando così *memory leaks* e non occupando memoria inutilmente, come mostrato nella classe `CaptureDeviceInput` del Listato 3.1.

Listato 3.1: Esempio di classe Java nella quale viene mantenuta la referenza ad un'oggetto nativo del QTKit

```
1 public class CaptureDeviceInput  {
2   /* ... */
3   private long objref_QTCaptureDeviceInput;
4
5   public CaptureDeviceInput(final CaptureDevice device) {
6     /* ... */
7     this.objref_QTCaptureDeviceInput = this.createQTCaptureDeviceInput(device);
8   }
9
```

```
10   protected void finalize ()  {
11             this.destructQTCaptureDeviceInput();
12   }
13
14   private native long createQTCaptureDeviceInput(final CaptureDevice device);
15   private native void destructQTCaptureDeviceInput();
16 }
```

Listato 3.2: Implementazione nativa della classe QTCaptureDeviceInput del Listato 3.1

```
1  JNIEXPORT jlong JNICALL
2  Java_it_unitn_science_JCQLibrary_CaptureDeviceInput_createQTCaptureDeviceInput
3  (JNIEnv *env, jobject caller, jobject obj) {
4    QTCaptureDeviceInput *nativeCaptureDeviceInput = NULL;
5
6  JNF_COCOA_ENTER(env);
7    /* ... */
8    // alloc and init the native object
9    nativeCaptureDeviceInput = [[QTCaptureDeviceInput alloc] initWithDevice:
               nativeCaptureDevice];
10 JNF_COCOA_EXIT(env);
11
12   // return the native object's pointer as reference
13   return (jlong)nativeCaptureDeviceInput;
14 }
15
16 JNIEXPORT void JNICALL
           Java_it_unitn_science_JCQLibrary_CaptureDeviceInput_destructQTCaptureDeviceInput

17 (JNIEnv *env, jobject caller) {
18 JNF_COCOA_ENTER(env);
19   // fetch native object from the saved reference
20   static JNF_CLASS_CACHE(jc_CaptureDeviceInput, "it/unitn/science/JCQLibrary/
               CaptureDeviceInput");
21   static JNF_MEMBER_CACHE(jm_CaptureDeviceInput_objref_QTCaptureDeviceInput,
               jc_CaptureDeviceInput, "objref_QTCaptureDeviceInput", "J");
22   jlong objref = JNFGetLongField(env, caller,
               jm_CaptureDeviceInput_objref_QTCaptureDeviceInput);
23   QTCaptureDeviceInput *nativeCaptureDeviceInput = (QTCaptureDeviceInput *)objref;
24
25   // release native object
26   [nativeCaptureDeviceInput release];
27 JNF_COCOA_EXIT(env);
28 }
```

3.2 Interazione interfaccia utente Java - Nativa

Oltre alla semplice condivisione di dati, JNI può essere usato anche per accedere alle risorse dell'interfaccia utente – User Interface (UI) – della piattaforma sottostante. Ci sono una serie di meccanismi per l'integrazione tra l'UI di Java e Cocoa utilizzando JNI.

Apple per comunicare con la propria interfaccia grafica di Cocoa offre il **CocoaComponent**. Il CocoaComponent è una classe astratta che permette l'incapsulamento di una *view* nativa di Cocoa (`NSView` o sua *subclass*[2]) all'interno di un componente grafico Java. CocoaComponent permette il passaggio di eventi e messaggi da AWT al componente Cocoa incapsulato in esso grazie all'uso della libreria JNI.
Per creare il proprio CocoaComponent bisogna:

- Estendere la classe Java `com.apple.eawt.CocoaComponent`.
- Implementare lato Java i metodi: `createNSView` e `createNSViewLong` del proprio CocoaComponent. A seconda dell'architettura del sistema operativo quando il CocoaComponent verrà creato per essere aggiunto alla gerarchia dei componenti grafici, uno di questi metodi verrà automaticamente invocato. Apple raccomanda l'implementazione solamente del costruttore `createNSViewLong`, che ritorna il puntatore come intero lungo di Java in 64 bit, e nel `createNSView` semplicemente ritornare il risultato della chiamata a `createNSViewLong` facendo un *casting* ad un intero normale Java (per il supporto a 32 bit).

```
1 // Instantiate the NSView on the native side and return it as a long
2 public native long createNSViewLong();
3
4 // Deprecated; just cast the correct createNSViewLong implementation
5 public int createNSView() {
6   return (int)createNSViewLong();
7 }
```

- Implementare la *view* nativa in Cocoa ereditando da `NSView`. Questo deciderà esattamente come sarà il proprio CocoaComponent e le funzionalità

[2]classe che ne eredita i metodi e le proprietà, estendendola.

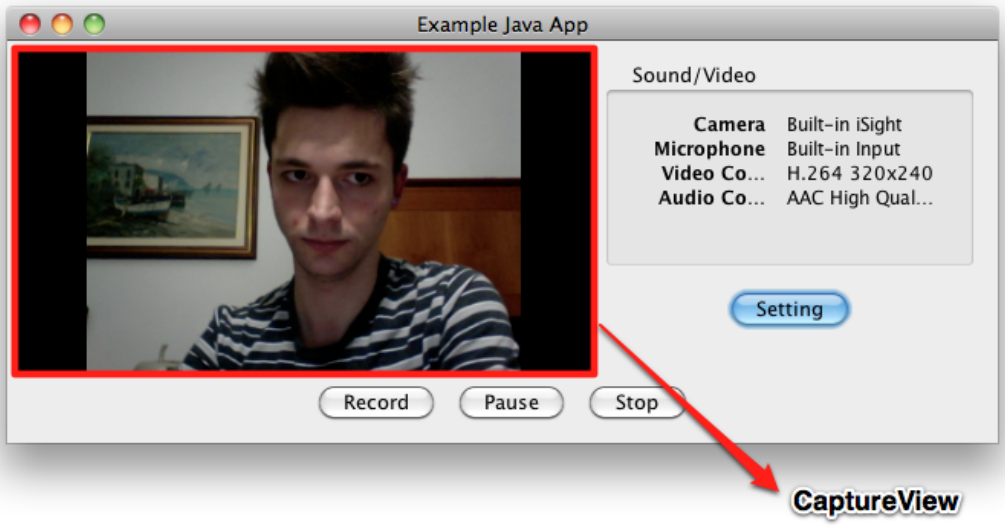

Figura 3.1: CaptureView realizzata tramite CocoaComponent per fornire l'anteprima video di cattura.

che offrirà: potrebbe essere una `WebView` per la visualizzazione di pagine web all'interno dell'applicazione, o qualsiasi altro oggetto basato su una `NSView` come nel nostro caso, in cui per fornire l'anteprima video è stata usata la `QTCaptureView` (vedi pag. 18) come mostrato in Figura 3.1.

- Implementare nativamente in JNI il metodo `createNSViewLong`. Si dovrà istanziare il proprio oggetto NSView e ritornarne il puntatore in Java come `jlong`.

```
1 JNIEXPORT jlong JNICALL
2 Java_com_mycompany_MyCocoaComponent_createNSViewLong
3 (JNIEnv *env, jobject caller) {
4   MyView *newView = [[[MyView alloc] init] autorelease];
5   return (jlong)newView;
6 }
```

Scambio messaggi

CocoaComponent prevede l'utilizzo del metodo `sendMessagge` per la comunicazione da Java a Cocoa, l'istruzione prevede il passaggio di un codice identificativo definito dallo sviluppatore e può accettare un oggetto come parametro opzionale.

L'utilizzo del metodo `sendMessagge` permette il *mapping* dei metodi della classe Java all'oggetto nativo inglobato nel CocoaComponent. Automaticamente quando si invoca un `sendMessagge` da Java a lato nativo viene chiamato il metodo `awtMessage:message:env` in cui dovranno trovarsi le istruzioni che implementano effettivamente la funzione desiderata, questo metodo è astratto e è definito nel protocollo Cocoa `AWTCocoaComponent` e all'interno di esso è possibile ottenere il codice identificativo del messaggio con cui è stato chiamato e l'eventuale parametro opzionale passato.

Listato 3.3: "CaptureView.java" - Esempio CocoaComponent per l'anteprima video usando la QTCaptureView di QTKit

```
1  package it.unitn.science.JCQLibrary;
2
3  import java.awt.*;
4
5  public class CaptureView extends com.apple.eawt.CocoaComponent {
6
7    static {
8      // ensure native JNI library is loaded
9      System.loadLibrary("JCQ");
10   }
11
12   final static Dimension MIN_SIZE   = new Dimension(376, 212);
13   final static Dimension PREF_SIZE  = new Dimension(376, 212);
14   final static Dimension MAX_SIZE   = new Dimension(500, 282);
15
16   // Constants for sending messages to the underlying NSView
17   final static int SESSION_MESSAGE  = 1;
18
19   private CaptureSession captureSession = null;
20
21   public CaptureSession getCaptureSession() {
22     return this.captureSession;
23   }
24
25   /*
26    * sendMessage wrapper methods
27    */
28
29   public void setCaptureSession(CaptureSession session) {
30     this.captureSession = session;
31     sendMessage(SESSION_MESSAGE, this.captureSession);
32   }
33
34   /*
```

```
35     *  Implementation of CocoaComponent abstracts
36     */
37
38   // Instantiate the NSView on the native side and return it as a long
39   public native long createNSViewLong();
40
41   // Deprecated; just cast the correct createNSViewLong implementation
42   public int createNSView() {
43     return (int)createNSViewLong();
44   }
45
46   public Dimension getPreferredSize() {
47     return PREF_SIZE;
48   }
49
50   public Dimension getMinimumSize () {
51     return MIN_SIZE;
52   }
53
54   public Dimension getMaximumSize() {
55     return MAX_SIZE;
56   }
57 }
```

Listato 3.4: "CaptureViewNative.h" - Esempio CocoaComponent per l'anteprima video usando la QTCaptureView di QTKit

```
1 #import <Cocoa/Cocoa.h>
2 #import <QTKit/QTCaptureView.h>
3 #import <JavaVM/AWTCocoaComponent.h>
4
5
6 @interface CaptureViewNative : QTCaptureView <AWTCocoaComponent> {}
7 @end
```

Listato 3.5: "CaptureViewNative.m" - Esempio CocoaComponent per l'anteprima video usando la QTCaptureView di QTKit

```
1  #import "CaptureViewNative.h"
2
3  #import <JavaNativeFoundation/JavaNativeFoundation.h>
4  #import "it_unitn_science_JCQLibrary_CaptureView.h"
5
6  /*
7   * Class:     Java_it_unitn_science_JCQLibrary_CaptureView
8   * Method:    nativeCreateNSViewLong
9   * Signature: ()J
10  */
11 JNIEXPORT jlong JNICALL
          Java_it_unitn_science_JCQLibrary_CaptureView_createNSViewLong
12 (JNIEnv *env, jobject caller) {
13   return (jlong)[[[CaptureViewNative alloc] init] autorelease];
14 }
15
16 @implementation CaptureViewNative
17
18
19 - (void)awtMessage:(jint)messageID message:(jobject)message  env:(JNIEnv*)env
20 {
21   switch (messageID) {
22     case it_unitn_science_JCQLibrary_CaptureView_SESSION_MESSAGE:
23       static JNF_CLASS_CACHE(jc_CaptureSession, "it/unitn/science/JCQLibrary/
                  CaptureSession");
24       static JNF_MEMBER_CACHE(jm_CaptureSession_objref_QTCaptureSession,
                   jc_CaptureSession, "objref_QTCaptureSession", "J");
25       QTCaptureSession *captureSession = nil;
26       if (message != NULL) {
27         jlong objref_QTCaptureSession = JNFGetLongField(env, message,
                    jm_CaptureSession_objref_QTCaptureSession);
28         captureSession = (QTCaptureSession *)objref_QTCaptureSession;
29       }
30       [self setCaptureSession:captureSession];
31       break;
32     default:
33       break;
34   }
35 }
36
37 @end
```

3.3 Compatibilità con QuickTime for Java

La nuova libreria JCQ purtroppo non è retrocompatibile[3] con le applicazioni che utilizzano QTJava – ad esempio LODE. Per questo motivo il codice sorgente dei programmi deve essere ripreso e modificato dai programmatori per renderlo compatibile con JCQ.

Può sembrare un controsenso, in quanto ambedue le librerie hanno origine comune: l'architettura QuickTime, ed entrambe le loro API sono definite e scritte con il linguaggio di programmazione Java, ma la natura della loro struttura è molto diversa. Infatti, la prima si appoggia direttamente all'API C di QuickTime e offre la maggior parte delle funzionalità rese disponibili nell'architettura di QuickTime; la seconda, invece, sfrutta il *subset*[4] "Capture" di QTKit che gestisce i dispositivi digitali, l'acquisizione e l'anteprima audio/video, utilizzandone gli oggetti per richiamarne nativamente i metodi e le proprietà lato Java, grazie a JNI.

Nello sviluppo e nella progettazione di QTJava, Apple si è basata sulle strutture dato, sulle procedure e sulle funzioni C di QuickTime e ha avuto una certa libertà nel design e nella creazione degli oggetti Java che formano il *wrapper*. Al contrario, essendo QTKit un *framework* Cocoa/Objective C e quindi OOP e ad un livello di astrazione superiore alle strutture C di QuickTime, nella progettazione di JCQ non è stato possibile mantenere la compatibilità con i metodi definiti in QTJava. Dovendone comunque utilizzare gli oggetti e i loro metodi per fornire le funzionalità multimediali, si è scelto, quindi, di mantenere anche la stessa struttura dell'API di QTKit, e quindi tranne alcuni oggetti e strutture *ad hoc*, le classi Java di JCQ hanno struttura, nome[5] e funzionalità pressoché uguali.

Fortunatamente JCQ risulta più semplice all'uso rispetto QTJava quindi la reimplemtazione da parte dei programmatori non è un grave problema e non dovrebbero incontrare alcuna complicazione. Per semplificarne ulteriormente l'uso è stato aggiunto un semplice oggetto: `JCQFactory` che predispone e comprende

[3] una libreria software è detta retrocompatibile se tutte le applicazioni che utilizzavano le versioni precedenti continuano a funzionare anche con la nuova versione.

[4] sottoinsieme, parte.

[5] Apple solitamente definisce i nomi degli oggetti all'interno dei propri framework inserendo un acronimo come prefisso per richiamarne l'appartenenza a quel determinato framework. Nel QTKit essi, infatti, hanno il prefisso "QT", mentre non compaiono nella nomenclatura data alle corrispettive classi Java di JCQ.

tutte le funzionalità base evitando l'uso diretto dei componenti alla base della libreria. La libreria viene rilasciata, inoltre, con la propria documentazione in Javadoc[10] – presente anche in Appendice A.

Capitolo 4

Conclusioni

Indice

Siamo infine giunti alla parte finale di questa relazione, in cui saranno presi in considerazione gli obiettivi raggiunti e mancati, le possibili estensioni future di interesse e le considerazioni generali su tutto il progetto.

4.1 Risultati ottenuti

JCQ soddisfa le funzionalità richieste e che erano state prefissate per la sua realizzazione, essa infatti permette: l'acquisizione digitale di audio/video da dispositivi digitali, quali videocamere, *iSight* e altre *webcam* e microfoni, spesso integrati nei moderni portatili e computer; e nel caso si desideri può offrire l'anteprima a video del processo di elaborazione del flusso di dati visivi, tramite il componente grafico specifico Java; infine permette l'esportazione in formato compresso o *raw* per una postelaborazione, come lo stesso QuickTime offre. Non è stato possibile mantenere il supporto a QTJava e quindi si è persa la compatibilità di tutti quei programmi

che la utilizzano contrariamente a quanto si era inizialmente prestabilito; lo stesso LODE necessita delle modifiche nelle istruzioni e nelle parti che utilizzano QTJava, in modo da integrarsi e supportare la nuova libreria JCQ.

Le funzionalità seppur basilari sono ottimizzate grazie allo sfruttamento, tramite JNI, dell'ottimo *framework* nativo: QTKit. L'API realizzata rappresenta una base per l'impiego nello sviluppo di applicazioni Java in sostituzione alla ormai superata QTJava per quelle funzionalità multimediali legate all'acquisizione.

L'API realizzata e i suoi codici sorgenti verranno rilasciati come open source e quindi sarà possibile estenderla, aggiungendo nuove funzionalità, continuarne il supporto e migliorarla.

4.2 Sviluppi futuri

Alcune idee per eventuali sviluppi futuri riguardano argomenti trattati in questa relazione e che non sono stati implementati per mancanza di tempo. Tra questi troviamo l'aggiunta delle restanti funzionalità di riproduzione, modifica, importazione e composizione che lo stesso QTKit offre.

Ulteriore miglioramento futuro sarebbe quello di offrire il supporto e la compatibilità anche a sistemi operativi diversi da Mac OS X, nel qual caso si dovrebbe provvedere ad una nuova implementazione della libreria nativa, sempre, tramite JNI per il nuovo sistema operativo. Il supporto a Microsoft Windows dovrebbe essere piuttosto semplice in quanto, se pur non esiste QTKit, sono disponibili le *ddl* con cui Apple fornisce il supporto all'API C di QuickTime; in Linux invece non è presente alcuna forma e supporto a QuickTime, quindi bisognerebbe svolgere una ricerca di tutta quella serie di librerie integrate nell'ambiente o esterne ma compatibili, che combinate assieme riescano a sostituire le funzionalità di QuickTime.

4.2.1 Java Media Framework

Java prevede il Java Media Framework (JMF): una libreria composta, fondamentalmente, da un insieme di API che consente, in modo versatile e relativamente semplice, di interfacciarsi con l'avvincente mondo dei media. L'architettura del JMF prevede l'utilizzo di una specifica implementazione, che espone alle applicazioni un

insieme di interfacce standard raggruppate nella sua API e si occupa di mappare a "run-time" i metodi opportuni per "pilotare" quelli specifici implementati al di sotto di esso. Sul web sono disponibili molte implementazioni di JMF, e al di fuori di quella che la Oracle rilascia assieme al *framework* stesso, ognuna è gestita in modo trasparente al programmatore. Con l'estensione di JCQ a tutte le funzionalità di QuickTime, un ulteriore step successivo potrebbe essere quello di usare il JCQ come implementazione reale delle specifiche standard di JMF. Questo renderebbe l'uso da parte dei programmatori e l'integrazione negli applicativi Java ancora più semplice e funzionale oltre ad aumentare le possibilità dei formati e dei file manipolabili di JMF sfruttando QuickTime.

Appendice A

Javadoc

Hierarchy For Package it.unitn.science.JCQLibrary

Class Hierarchy

- java.lang.Object
  - it.unitn.science.JCQLibrary.**CaptureDevice**
  - it.unitn.science.JCQLibrary.**CaptureDeviceInput**
  - it.unitn.science.JCQLibrary.**CaptureMovieFileOutput**
  - it.unitn.science.JCQLibrary.**CaptureSession**
  - java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - java.awt.Canvas (implements javax.accessibility.Accessible)
      - com.apple.eawt.CocoaComponent
        - it.unitn.science.JCQLibrary.**CaptureView**
  - it.unitn.science.JCQLibrary.**CompressionOptions**
  - it.unitn.science.JCQLibrary.**JCQFactory**
  - java.lang.Throwable (implements java.io.Serializable)
    - java.lang.Exception
      - it.unitn.science.JCQLibrary.**CompressionOptions.InvalidCompressionException**

Enum Hierarchy

- java.lang.Object
  - java.lang.Enum<E> (implements java.lang.Comparable<T>, java.io.Serializable)
    - it.unitn.science.JCQLibrary.**MediaType**

Package it.unitn.science.JCQLibrary

Class Summary	
CaptureDevice	This class represents an available capture device.
CaptureDeviceInput	This class represents the input source for media devices, such as cameras and microphones.
CaptureMovieFileOutput	This class represents an output destination for `CaptureSession` that writes captured media to QuickTime movie files.
CaptureSession	This class is the primary interface for capturing media streams.
CaptureView	This is an AWT graphical component that displays a video preview of a capture session.
CompressionOptions	This class represents a set of compression options for a particular type of media.
JCQFactory	The JCQFactory class defines the simple way for capturing and previewing from a video and a microphone device.

Enum Summary	
MediaType	This enum represents a particular type of media.

Exception Summary

it.unitn.science.JCQLibrary

Class CaptureDevice

```
java.lang.Object
  extended by  it.unitn.science.JCQLibrary.CaptureDevice
```

```
public class CaptureDeviceextends java.lang.Object
```

This class represents an available capture device. Each instance of CaptureDevice corresponds to a capture device that is connected or has been previously connected to the user‘s computer during the lifetime of the application. An instance can be created using the `CaptureDevice(String)` class method. A `List` of all currently connected devices can also be obtained using the `getInputDevices()` class method.

Devices can provide one or more stream of a given media type. Applications can search for devices that provide media of a specific type using the `getInputDevicesWithMediaType(MediaType)` and `getDefaultInputDeviceWithMediaType(MediaType)` class methods.

Since:

1.5

Constructor Summary

`CaptureDevice``(java.lang.String uniqueID)` Initializes a newly created CaptureDevice object with the identifier device UID.

Method Summary	
void	**close**() Releases application control over the device acquired in the open() method.
boolean	**equals**(java.lang.Object k)
static CaptureDevice	**getDefaultInputDeviceWithMediaType**(MediaType mediatype) Returns a CaptureDevice instance for the default device connected to the user's system of the given media type.
static java.util.List<CaptureDevice>	**getInputDevices**() Returns a list of devices currently connected to the computer that can be used as input sources.
static java.util.List<CaptureDevice>	**getInputDevicesWithMediaType**(MediaType mediatype) Returns a list of input devices currently connected to the computer that send a stream with the given media type.
java.lang.String	**getLocalizedDisplayName**() Returns a localized human-readable name for the receiver's device.
java.lang.String	**getUniqueID**() Returns the unique ID of the receiver's device.
int	**hashCode**()
boolean	**hasMediaType**(MediaType mediatype) Returns whether the receiver sends a stream with the given media type.
boolean	**isOpen**() Returns true if the device is open in the current application.
boolean	**open**() Attempts to give the application control over the device so that it can be used for capture.
java.lang.String	**toString**()

Methods inherited from class java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

CaptureDevice

public **CaptureDevice**(java.lang.String uniqueID)

Initializes a newly created CaptureDevice object with the identifier device UID. Every capture device available to the computer is assigned a unique identifier that persists on one computer across device connections and disconnections, as well as across reboots of the computer. This method can be used to recall or track the status of a specific device, even if it has been disconnected.

Parameters:

`uniqueID` - the unique identifier of the device instance to be returned

Method Detail

getDefaultInputDeviceWithMediaType

```
public static CaptureDevice getDefaultInputDeviceWithMediaType(MediaType mediatype)
```

Returns a CaptureDevice instance for the default device connected to the user's system of the given media type. This method returns the default device of the given media type connected to the user's system. For example, for Mediatype.SOUND, this method will return the default sound input device selected in the Sound Preference Pane. If there is no device for the given media type, this method will return `null`.

Parameters:

`mediatype` - the media type, such as Mediatype.VIDEO, Mediatype.SOUND, or Mediatype.MUXED, supported by the returned device

Returns:

the default device with the given media type on the user's system, or `null` if no device with that media type exists

See Also:

`MediaType`

---

getInputDevices

```
public static java.util.List<CaptureDevice> getInputDevices()
```

Returns a list of devices currently connected to the computer that can be used as input sources. This method queries the device system and builds a list of CaptureDevice instances for input devices currently connected and available for capture. The returned list contains all devices that are available when the method is called.

Returns:

returns a `List` of CaptureDevice instances for each connected device. If there are no available devices, the returned array will be empty

See Also:

`List`

---

getInputDevicesWithMediaType

```
public static java.util.List<CaptureDevice> getInputDevicesWithMediaType(MediaType mediatype)
```

Returns a list of input devices currently connected to the computer that send a stream with the given media type. This method queries the device system and builds a list of CaptureDevice instances for input devices that are currently connected and output streams of the given media type.

Parameters:

`mediatype` - the media type, such as Mediatype.VIDEO, Mediatype.SOUND, or Mediatype.MUXED, supported by the returned device

Returns:

returns a `List` of CaptureDevice instances for each connected device with the given media type. If there are no available devices, the returned list will be empty

See Also:

`MediaType`, `List`

---

hasMediaType

```
public boolean hasMediaType(MediaType mediatype)
```

Returns whether the receiver sends a stream with the given media type. This method queries the device system and builds a list of CaptureDevice instances for input devices that are currently connected and output streams of the given media type.

Parameters:

`mediatype` - the media type, such as Mediatype.VIDEO, Mediatype.SOUND, or Mediatype.MUXED, supported by the returned device

Returns:

returns `true` if the device outputs the given media type, `false` otherwise

See Also:

`MediaType`

---

open

```
public boolean open()
```

Attempts to give the application control over the device so that it can be used for capture. This method attempts to open the device for control by the current application. If the device is connected and no other processes have exclusive control over it, then the application starts using the device immediately, taking exclusive control of it if necessary. Otherwise, this method returns `false`. Applications that call open() should also call the `close()` method to relinquish access to the device when it is no longer needed. Multiple calls to this method can be nested. Each call to this method must be matched by a call to close. Applications that capture from a device using CaptureDeviceInput must call this method before creating the CaptureDeviceInput to be used with the device. If a device is disconnected or turned off while it is open, it will be closed automatically.

Returns:

returns `true` if the device was opened successfully; otherwise, `false`

See Also:

`CaptureDeviceInput`

---

isOpen

```
public boolean isOpen()
```

Returns `true` if the device is open in the current application. This method attempts to open the device for control by the current application. If the device is connected and no other processes have exclusive control over it, then the application starts using the device immediately, taking exclusive control of it if necessary. Otherwise, this method returns `false`. Applications that call open() should also call the `close()` method to relinquish access to the device when it is no longer needed. Multiple calls to this method can be nested. Each call to this method must be matched by a call to close. Applications that capture from a device using CaptureDeviceInput must call this method before creating the CaptureDeviceInput to be used with the device. If a device is disconnected or turned off while it is open, it will be closed automatically.

Returns:

returns `true` if the device was previously opened by the receiver's `open()` method. Returns `false` otherwise

---

close

```
public void close()
```

Releases application control over the device acquired in the `open()` method. This method should be called to match each invocation of `open()` when an application no longer needs to use a device for capture. If a device is disconnected or turned off while it is open it will be closed automatically.

---

getUniqueID

```
public java.lang.String getUniqueID()
```

Returns the unique ID of the receiver's device. The unique identifier returned by this method is persistent on one computer across device connections and disconnections, as well as across reboots of the computer. It can be passed to the `CaptureDevice(String)` class method to get the CaptureDevice instance for the device with that unique identifier.

Returns:

the unique identifier of the device corresponding to the receiver.

---

getLocalizedDisplayName

```
public java.lang.String getLocalizedDisplayName()
```

Returns a localized human-readable name for the receiver's device. This method can be used when displaying the name of a capture device in the user interface.

Returns:

The localized name of the receiver's device

toString

```
public java.lang.String toString()
```

Overrides:
`toString` in class `java.lang.Object`

---

equals

```
public boolean equals(java.lang.Object k)
```

Overrides:
`equals` in class `java.lang.Object`

---

hashCode

```
public int hashCode()
```

Overrides:
`hashCode` in class `java.lang.Object`

---

it.unitn.science.JCQLibrary

Class CaptureDeviceInput

```
java.lang.Object
  extended by  it.unitn.science.JCQLibrary.CaptureDeviceInput
```

```
public class CaptureDeviceInputextends java.lang.Object
```

This class represents the input source for media devices, such as cameras and microphones. Instances of CaptureDeviceInput are input sources for `CaptureSession` that provide media data from devices connected to the computer. Devices used with CaptureDeviceInput can be found using the `CaptureDevice` class. A `CaptureDevice` must be successfully opened using the `CaptureDevice.open()` method before being used in a CaptureDeviceInput.

Since:
1.5

Constructor Summary

`CaptureDeviceInput``(CaptureDevice device)` Initializes a newly created CaptureDeviceInput object associated with the given device.	

Method Summary

boolean	**`equals`**`(java.lang.Object k)`
protected void	**`finalize`**`()` Destruct also the native object with a `release` associated with the receiver.
CaptureDevice	**`getDevice`**`()` Returns the device associated with the receiver.
int	**`hashCode`**`()`

Methods inherited from class java.lang.Object
`clone, getClass, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

CaptureDeviceInput

```
public CaptureDeviceInput(CaptureDevice device)
```

Initializes a newly created CaptureDeviceInput object associated with the given device.

Parameters:

`device` - a CaptureDevice object for the device to be associated with the receiver. The device must have been previously opened using the `CaptureDevice.open()` method.

Method Detail

getDevice

```
public CaptureDevice getDevice()
```

Returns the device associated with the receiver.

Returns:

If there is a device associated with the receiver, returns a corresponding instance of `CaptureDevice`. Otherwise returns `null`.

See Also:

`CaptureDevice`

---

finalize

```
protected void finalize()
```

Destruct also the native object with a `release` associated with the receiver.

Overrides:

`finalize` in class `java.lang.Object`

---

equals

```
public boolean equals(java.lang.Object k)
```

Overrides:

`equals` in class `java.lang.Object`

---

hashCode

```
public int hashCode()
```

Overrides:

`hashCode` in class `java.lang.Object`

---

it.unitn.science.JCQLibrary

Class CaptureMovieFileOutput

```
java.lang.Object
  extended by  it.unitn.science.JCQLibrary.CaptureMovieFileOutput
```

```
public class CaptureMovieFileOutputextends java.lang.Object
```

This class represents an output destination for `CaptureSession` that writes captured media to QuickTime movie files. A CaptureMovieFileOutput instance writes the media captured by its connected capture session to QuickTime movie files.

Since:

1.5

Constructor Summary

`CaptureMovieFileOutput()` Initializes a newly created CaptureMovieFileOutput object.	

Method Summary

protected void	**finalize()** Destruct also the native object with a `release` associated with the receiver.
CompressionOptions	**getAudioCompressionOptions()** Returns the audio compression of the file written to by the receiver.
CompressionOptions	**getVideoCompressionOptions()** Returns the video compression of the file written to by the receiver.
boolean	**isOpenCapturedFile()** Return if when the file written to by the receiver is closed have to be opened.
boolean	**isRecordingPaused()** Returns whether recording to the current output file is paused.
void	**pauseRecording()** Pauses recording to the current output file.
void	**recordToOutputFileURL**(java.lang.String path) Sets the file written to by the receiver, specifying where the sample buffer from a new file should be recorded.
void	**resumeRecording()** Resumes recording to the current output file after it was previously paused using `pauseRecording()`.
void	**setAudioCompressionOptions**(CompressionOptions compression) Sets the audio compression of the file written to by the receiver, specifying the desired audio compression.
void	**setOpenCapturedFile**(boolean flag) Sets if when the file written to by the receiver is closed have to be opened.
void	**setVideoCompressionOptions**(CompressionOptions compression) Sets the video compression of the file written to by the receiver, specifying the desired video compression.

Methods inherited from class java.lang.Object

`clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

CaptureMovieFileOutput

`public` **`CaptureMovieFileOutput()`**

Initializes a newly created CaptureMovieFileOutput object.

Method Detail

finalize

```
protected void finalize()
```

Destruct also the native object with a `release` associated with the receiver.

Overrides:
`finalize` in class `java.lang.Object`

---

recordToOutputFileURL

```
public void recordToOutputFileURL(java.lang.String path)
```

Sets the file written to by the receiver, specifying where the sample buffer from a new file should be recorded. The method sets the file path to which the receiver is currently writing output media. If a file at the given path already exists when capturing starts, the existing file is overwritten. If `null` is passed as the file path, the receiver will stop recording to any file. If this method is invoked while an existing output file was already being recorded, no media samples are discarded between the old file and the new file. The sample buffer currently in flight when this method is called will always be written to the new file.

Parameters:
`path` - object containing the path of the output file, or `null` if the receiver should not record to any file.

---

isRecordingPaused

```
public boolean isRecordingPaused()
```

Returns whether recording to the current output file is paused. This method returns whether recording to the file has been previously paused using the `pauseRecording()` method. When a recording is paused, captured samples are not written to the output file, but new samples can be written to the same file in the future by calling `resumeRecording()`.

Returns:
returns `true` if recording to the current output file is paused and returns `false` otherwise

---

pauseRecording

```
public void pauseRecording()
```

Pauses recording to the current output file. This method causes the receiver to stop writing captured samples to the current output file, but leaves the file open so that samples can be written to it in the future, when `resumeRecording()` is called. This allows clients to record multiple media segments that are not contiguous in time to a single file.

When clients stop recording or change files using `recordToOutputFileURL(java.lang.String)` or recording automatically stops due to an error condition while recording is paused, the output file will be finished and closed normally without requiring a matching call to `resumeRecording()`. When there is no current output file, or when recording is already paused, this method does nothing.

---

resumeRecording

```
public void resumeRecording()
```

Resumes recording to the current output file after it was previously paused using `pauseRecording()`. This method causes the receiver to resume writing captured samples to the current output file, after recording was previously paused using `pauseRecording()`. This allows clients to record multiple media segments that are not contiguous in time to a single file. When there is no current output file, or when recording is not paused, this method does nothing.

---

setVideoCompressionOptions

```
public void setVideoCompressionOptions(CompressionOptions compression)
```

Sets the video compression of the file written to by the receiver, specifying the desired video compression. The method sets the video compression of the file to which the receiver is currently writing output media. If the passed compression is not a kind of video one it does nothing.

Parameters:

`compression` - object containing the desired video compression of the output file.

See Also:

`CompressionOptions`

---

setAudioCompressionOptions

```
public void setAudioCompressionOptions(CompressionOptions compression)
```

Sets the audio compression of the file written to by the receiver, specifying the desired audio compression. The method sets the audio compression of the file to which the receiver is currently writing output media. If the passed compression is not a kind of sound one it does nothing.

Parameters:

`compression` - object containing the desired audio compression of the output file.

See Also:

`CompressionOptions`

---

setOpenCapturedFile

```
public void setOpenCapturedFile(boolean flag)
```

Sets if when the file written to by the receiver is closed have to be opened. The method sets if when the file written to by the receiver is closed by a `recordToOutputFileURL(java.lang.String)` have to be opened with the default

player application (eg. QuickTime).

Parameters:
`flag` - if `true` the file will be opened if `false` not

---

getVideoCompressionOptions

```
public CompressionOptions getVideoCompressionOptions()
```

Returns the video compression of the file written to by the receiver.

Returns:
the video compression of the file to which the receiver is currently writing output media.
See Also:
`CompressionOptions`

---

getAudioCompressionOptions

```
public CompressionOptions getAudioCompressionOptions()
```

Returns the audio compression of the file written to by the receiver.

Returns:
the audio compression of the file to which the receiver is currently writing output media.
See Also:
`CompressionOptions`

---

isOpenCapturedFile

```
public boolean isOpenCapturedFile()
```

Return if when the file written to by the receiver is closed have to be opened. The method return if when the file written to by the receiver is closed by a `recordToOutputFileURL(java.lang.String)` have to be opened with the default player application (eg. QuickTime).

Returns:
`true` if the file will be opened, `false` otherwise

---

it.unitn.science.JCQLibrary

Class CaptureSession

```
java.lang.Object
  extended by  it.unitn.science.JCQLibrary.CaptureSession
```

```
public class CaptureSessionextends java.lang.Object
```

This class is the primary interface for capturing media streams. A CaptureSession instance provides an interface for connecting capture input sources, `CaptureDeviceInput` to output destinations, `CaptureMovieFileOutput` and preview of the `CaptureView`. In addition to managing the connections between inputs and outputs, instances of CaptureSession also manage when a capture is running.

Since:

1.5

Constructor Summary

`CaptureSession()` Initializes a newly created CaptureSession object.	

Method Summary	
boolean	**addInput**(CaptureDeviceInput input) Adds an input to the receiver.
boolean	**addOutput**(CaptureMovieFileOutput output) Adds an output to the receiver.
protected void	**finalize**() Destruct also the native object with a `release` associated with the receiver.
java.util.List<CaptureDeviceInput>	**getInputs**() Returns a list of inputs connected to the receiver.
java.util.List<CaptureMovieFileOutput>	**getOutputs**() Returns a list of outputs connected to the receiver.
boolean	**isRunning**() Returns whether the receiver is running.
void	**removeInput**(CaptureDeviceInput input) Removes an input from the receiver.
void	**removeOutput**(CaptureMovieFileOutput output) Removes an ouput from the receiver.
void	**startRunning**() Tells the receiver to start capturing data from its inputs and sending data to its outputs.
void	**stopRunning**() Tells the receiver to stop capturing data from its inputs and sending data to its outputs.

Methods inherited from class java.lang.Object
clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

CaptureSession

`public` **CaptureSession**()

Initializes a newly created CaptureSession object.

Method Detail

finalize

```
protected void finalize()
```

Destruct also the native object with a `release` associated with the receiver.

Overrides:
`finalize` in class `java.lang.Object`

---

isRunning

```
public boolean isRunning()
```

Returns whether the receiver is running. When a CaptureSession is running, it continuously reads media from its inputs and sends it to those outputs currently accepting data. When data does not need to be sent to file outputs, previews, and other outputs, capture sessions should not be running so that the overhead from capturing not affect application performance. By default, capture sessions are not running.

Returns:
returns `true` if the receiver is running, `false` otherwise.

---

startRunning

```
public void startRunning()
```

Tells the receiver to start capturing data from its inputs and sending data to its outputs. When a CaptureSession is running, it continuously reads media from its inputs and sends it to those outputs currently accepting data. When data does not need to be sent to file outputs, previews, and other outputs, capture sessions should not be running so that the overhead from capturing not affect application performance. By default, capture sessions are not running.

---

stopRunning

```
public void stopRunning()
```

Tells the receiver to stop capturing data from its inputs and sending data to its outputs. When a CaptureSession is running, it continuously reads media from its inputs and sends it to those outputs currently accepting data. When data does not need to be sent to file outputs, previews, and other outputs, capture sessions should not be running so that the overhead from capturing not affect application performance. By default, capture sessions are not running.

---

getInputs

```
public java.util.List<CaptureDeviceInput> getInputs()
```

Returns a list of inputs connected to the receiver. A capture session can have one or more input sources, which are instances of `CaptureDeviceInput`.

Returns:
returns a `List` of `CaptureDeviceInput` instances
See Also:
`List`

---

getOutputs

```
public java.util.List<CaptureMovieFileOutput> getOutputs()
```

Returns a list of outputs connected to the receiver. A capture session can have one or more output sources, which are instances of `CaptureMovieFileOutput`.

Returns:
returns a `List` of `CaptureMovieFileOutput` instances
See Also:
`List`

---

addInput

```
public boolean addInput(CaptureDeviceInput input)
```

Adds an input to the receiver. This method adds a `CaptureDeviceInput` to the receiverâs list of inputs, adding each of its connections to the capture session as media sources. If there are any outputs already added to the receiver after an input is successfully added, each output creates an additional connection for each stream of media that it can read from the session. If an input is added successfully this method returns `true`. If an input is added more than once, this method does nothing and returns `true`. If an input cannot be added, this method returns `false`.

Parameters:
`input` - the capture device input to be connected to the receiver
Returns:
returns `true` if the input was added successfully, or has already been added to the receiver. Returns `false` if the input could not be added

---

addOutput

```
public boolean addOutput(CaptureMovieFileOutput output)
```

Adds an output to the receiver. This method adds a `CaptureMovieFileOutput` to the receiverâs list of outputs. After an output is successfully added to a session, it creates one connection for each stream of media that it can read from the session. If an output is added successfully this method returns `true`. If an output is added more than once, this method does nothing and returns `true`. If an output cannot be added, this method returns `false`.

Parameters:
`output` - the file output to be connected to the receiver
Returns:
returns `true` if the output was added successfully, or has already been added to the receiver. Returns `false` if the output could not be added

removeInput

```
public void removeInput(CaptureDeviceInput input)
```

Removes an input from the receiver. This method removes a `CaptureDeviceInput` added with `addInput(CaptureDeviceInput)`.

Parameters:

`input` - the input to be removed from the receiver

---

removeOutput

```
public void removeOutput(CaptureMovieFileOutput output)
```

Removes an ouput from the receiver. This method removes a `CaptureMovieFileOutput` added with `addOutput(CaptureMovieFileOutput)`.

Parameters:

`output` - the output to be removed from the receiver

---

it.unitn.science.JCQLibrary

Class CaptureView

```
java.lang.Object
  extended by  java.awt.Component
      extended by  java.awt.Canvas
          extended by  com.apple.eawt.CocoaComponent
              extended by  it.unitn.science.JCQLibrary.CaptureView
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

---

```
public class CaptureViewextends com.apple.eawt.CocoaComponent
```

This is an AWT graphical component that displays a video preview of a capture session. A CaptureView previews the video being processed by an instance of `CaptureSession`.

Since:

1.5

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class java.awt.Canvas
java.awt.Canvas.AccessibleAWTCanvas

Nested classes/interfaces inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent, java.awt.Component.BaselineResizeBehavior, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

CaptureView()	

Method Summary

int	**createNSView**()
long	**createNSViewLong**()
CaptureSession	**getCaptureSession**() Returns the capture session being previewed by the receiver.
java.awt.Dimension	**getMaximumSize**()
java.awt.Dimension	**getMinimumSize**()
java.awt.Dimension	**getPreferredSize**()
void	**setCaptureSession**(CaptureSession session) Sets the capture session to be previewed by the receiver.

Methods inherited from class com.apple.eawt.CocoaComponent
paint, sendMessage, update

Methods inherited from class java.awt.Canvas
addNotify, createBufferStrategy, createBufferStrategy, getAccessibleContext, getBufferStrategy

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener,
addHierarchyBoundsListener, addHierarchyListener,
addInputMethodListener, addKeyListener, addMouseListener,
addMouseMotionListener, addMouseWheelListener,
addPropertyChangeListener, addPropertyChangeListener,
applyComponentOrientation, areFocusTraversalKeysSet, bounds,
checkImage, checkImage, coalesceEvents, contains, contains,
createImage, createImage, createVolatileImage, createVolatileImage,
deliverEvent, disable, disableEvents, dispatchEvent, doLayout,
enable, enable, enableEvents, enableInputMethods,
firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange,
getAlignmentX, getAlignmentY, getBackground, getBaseline,
getBaselineResizeBehavior, getBounds, getBounds, getColorModel,
getComponentAt, getComponentAt, getComponentListeners,
getComponentOrientation, getCursor, getDropTarget,
getFocusCycleRootAncestor, getFocusListeners,
getFocusTraversalKeys, getFocusTraversalKeysEnabled, getFont,
getFontMetrics, getForeground, getGraphics,
getGraphicsConfiguration, getHeight, getHierarchyBoundsListeners,
getHierarchyListeners, getIgnoreRepaint, getInputContext,
getInputMethodListeners, getInputMethodRequests, getKeyListeners,
getListeners, getLocale, getLocation, getLocation,
getLocationOnScreen, getMouseListeners, getMouseMotionListeners,
getMousePosition, getMouseWheelListeners, getName, getParent,
getPeer, getPropertyChangeListeners, getPropertyChangeListeners,
getSize, getSize, getToolkit, getTreeLock, getWidth, getX, getY,
gotFocus, handleEvent, hasFocus, hide, imageUpdate, inside,
invalidate, isBackgroundSet, isCursorSet, isDisplayable,
isDoubleBuffered, isEnabled, isFocusable, isFocusCycleRoot,
isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet,
isLightweight, isMaximumSizeSet, isMinimumSizeSet, isOpaque,
isPreferredSizeSet, isShowing, isValid, isVisible, keyDown, keyUp,
layout, list, list, list, list, list, locate, location, lostFocus,
minimumSize, mouseDown, mouseDrag, mouseEnter, mouseExit,
mouseMove, mouseUp, move, nextFocus, paintAll, paramString,
postEvent, preferredSize, prepareImage, prepareImage, print,
printAll, processComponentEvent, processEvent, processFocusEvent,
processHierarchyBoundsEvent, processHierarchyEvent,
processInputMethodEvent, processKeyEvent, processMouseEvent,
processMouseMotionEvent, processMouseWheelEvent, remove,
removeComponentListener, removeFocusListener,
removeHierarchyBoundsListener, removeHierarchyListener,
removeInputMethodListener, removeKeyListener, removeMouseListener,
removeMouseMotionListener, removeMouseWheelListener, removeNotify,
removePropertyChangeListener, removePropertyChangeListener,
repaint, repaint, repaint, repaint, requestFocus, requestFocus,
requestFocusInWindow, requestFocusInWindow, reshape, resize,
resize, setBackground, setBounds, setBounds,
setComponentOrientation, setCursor, setDropTarget, setEnabled,
setFocusable, setFocusTraversalKeys, setFocusTraversalKeysEnabled,
setFont, setForeground, setIgnoreRepaint, setLocale, setLocation,
setLocation, setMaximumSize, setMinimumSize, setName,
setPreferredSize, setSize, setSize, setVisible, show, show, size,
toString, transferFocus, transferFocusBackward,
transferFocusUpCycle, validate

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

CaptureView

```
public CaptureView()
```

Method Detail

setCaptureSession

```
public void setCaptureSession(CaptureSession session)
```

Sets the capture session to be previewed by the receiver.

Parameters:
`session` - a `CaptureSession` instance to be used for the preview.

---

getCaptureSession

```
public CaptureSession getCaptureSession()
```

Returns the capture session being previewed by the receiver.

Returns:
a `CaptureSession` used for the preview.

---

createNSViewLong

```
public long createNSViewLong()
```

Overrides:
`createNSViewLong` in class `com.apple.eawt.CocoaComponent`

---

createNSView

```
public int createNSView()
```

Specified by:
`createNSView` in class `com.apple.eawt.CocoaComponent`

---

getPreferredSize

```
public java.awt.Dimension getPreferredSize()
```

Specified by:
`getPreferredSize` in class `com.apple.eawt.CocoaComponent`

---

getMinimumSize

```
public java.awt.Dimension getMinimumSize()
```

Specified by:
`getMinimumSize` in class `com.apple.eawt.CocoaComponent`

---

getMaximumSize

```
public java.awt.Dimension getMaximumSize()
```

Specified by:
`getMaximumSize` in class `com.apple.eawt.CocoaComponent`

---

it.unitn.science.JCQLibrary

Class CompressionOptions.InvalidCompressionException

```
java.lang.Object
  extended by  java.lang.Throwable
      extended by  java.lang.Exception
          extended by  it.unitn.science.JCQLibrary.CompressionOptions.InvalidCompressionException
```

All Implemented Interfaces:
java.io.Serializable

Enclosing class:
CompressionOptions

```
public class CompressionOptions.InvalidCompressionExceptionextends java.lang.Exception
```

Thrown when an application attempts to use a non valid CompressionOptions.

See Also:
Serialized Form

Constructor Summary

`CompressionOptions.InvalidCompressionException()` Constructs a InvalidCompressionException with no detail message.
`CompressionOptions.InvalidCompressionException(java.lang.String s)` Constructs a InvalidCompressionException with the specified detail message.

Method Summary

`java.lang.String`	**`getMessage()`** Return the receiver's message.

Methods inherited from class java.lang.Throwable
`fillInStackTrace, getCause, getLocalizedMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

CompressionOptions.InvalidCompressionException

```
public CompressionOptions.InvalidCompressionException()
```

Constructs a InvalidCompressionException with no detail message.

CompressionOptions.InvalidCompressionException

```
public CompressionOptions.InvalidCompressionException(java.lang.String s)
```

Constructs a InvalidCompressionException with the specified detail message.

Parameters:
`s` - the detail message

Method Detail

getMessage

```
public java.lang.String getMessage()
```

Return the receiver's message.

Overrides:
`getMessage` in class `java.lang.Throwable`
Returns:
the detail message

---

it.unitn.science.JCQLibrary

Class CompressionOptions

```
java.lang.Object
  extended by  it.unitn.science.JCQLibrary.CompressionOptions
```

```
public class CompressionOptionsextends java.lang.Object
```

This class represents a set of compression options for a particular type of media. CompressionOptions objects are used to describe compression options for different kinds of media. Compression options are created from presets keyed by a named identifier. Preset identifiers are described in the costant of the class.

Note that not all documented identifiers may be available for a given system configuration. Clients should always query for available identifiers first.

Since:
1.5

Nested Class Summary

class	**CompressionOptions.InvalidCompressionException** Thrown when an application attempts to use a non valid CompressionOptions.

Field Summary

static java.lang.String	**H264_120_VIDEO** Compresses video using the H.264 codec using medium bit-rate settings with dimensions no larger than 160x120.
static java.lang.String	**H264_240_VIDEO** Compresses video using the H.264 codec using medium bit-rate settings with dimensions no larger than 320x240.
static java.lang.String	**H264_480_VIDEO** Compresses video using the H.264 codec using medium bit-rate settings with dimensions no larger than 720x480.
static java.lang.String	**HQ_AAC_AUDIO** Compresses audio using the AAC codec at 64 kbps per channel.
static java.lang.String	**JPEG_VIDEO**
static java.lang.String	**LOSSLESS_ALAC_AUDIO** Compresses audio using the Apple Lossless codec.
static java.lang.String	**LOSSLESS_ANIMATION_VIDEO** Compresses video using the Animation codec at highest quality and color depth.
static java.lang.String	**LOSSLESS_INTERMEDIATE_VIDEO** Compresses video using the Apple Intermediate codec at lossless quality.
static java.lang.String	**MPEG4_120_VIDEO** Compresses video using the MPEG-4 codec using medium bit-rate settings with dimensions no larger than 160x120.
static java.lang.String	**MPEG4_240_VIDEO** Compresses video using the MPEG-4 codec using medium bit-rate settings with dimensions no larger than 320x240.
static java.lang.String	**MPEG4_480_VIDEO** Compresses video using the MPEG-4 codec using medium bit-rate settings with dimensions no larger than 720x480.
static java.lang.String	**VQ_AAC_AUDIO** Compresses audio using the AAC codec at 32 kbps per channel.

Constructor Summary

CompressionOptions(java.lang.String identifier) Initializes a newly created CompressionOptions object configured for the given identifier.

Method Summary	
boolean	**equals**(java.lang.Object k)
static java.util.List<CompressionOptions>	**getCompressionOptionsAvailableForMediaType**(MediaType mediaType) Returns the list of all possible and available compression options for the given media type on the userâs system.
java.lang.String	**getLocalizedCompressionOptionsSummary**() Returns a localized summary of the receiverâs compression options.
java.lang.String	**getLocalizedDisplayName**() Returns a short localized name describing the receiverâs compression options.
MediaType	**getMediaType**() Returns the media type on which the receiverâs compression options should be used.
int	**hashCode**()
java.lang.String	**toString**()

Methods inherited from class java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Field Detail

LOSSLESS_INTERMEDIATE_VIDEO

public static final java.lang.String **LOSSLESS_INTERMEDIATE_VIDEO**

Compresses video using the Apple Intermediate codec at lossless quality. This is appropriate for an intermediate format for media that requires further processing. Only available in 32-bit.

See Also:
Constant Field Values

LOSSLESS_ANIMATION_VIDEO

public static final java.lang.String **LOSSLESS_ANIMATION_VIDEO**

Compresses video using the Animation codec at highest quality and color depth. This is appropriate for an intermediate format for media that requires further processing.

See Also:
Constant Field Values

JPEG_VIDEO

```
public static final java.lang.String JPEG_VIDEO
```

See Also:
Constant Field Values

H264_120_VIDEO

```
public static final java.lang.String H264_120_VIDEO
```

Compresses video using the H.264 codec using medium bit-rate settings with dimensions no larger than 160x120. This is appropriate for delivery to low-bandwidth and low-capacity destinations.

See Also:
Constant Field Values

H264_240_VIDEO

```
public static final java.lang.String H264_240_VIDEO
```

Compresses video using the H.264 codec using medium bit-rate settings with dimensions no larger than 320x240. This is appropriate for delivery to medium-bandwidth and medium-capacity destinations.

See Also:
Constant Field Values

H264_480_VIDEO

```
public static final java.lang.String H264_480_VIDEO
```

Compresses video using the H.264 codec using medium bit-rate settings with dimensions no larger than 720x480. This is appropriate for delivery to medium and high-bandwidth and medium- and high-capacity destinations.

See Also:
Constant Field Values

MPEG4_120_VIDEO

```
public static final java.lang.String MPEG4_120_VIDEO
```

Compresses video using the MPEG-4 codec using medium bit-rate settings with dimensions no larger than 160x120. This is appropriate for delivery to low-bandwidth and low-capacity destinations. Only available in 32-bit.

See Also:
Constant Field Values

MPEG4_240_VIDEO

```
public static final java.lang.String MPEG4_240_VIDEO
```

Compresses video using the MPEG-4 codec using medium bit-rate settings with dimensions no larger than 320x240. This is appropriate for delivery to medium-bandwidth and medium-capacity destinations. Only available in 32-bit.

See Also:
Constant Field Values

MPEG4_480_VIDEO

```
public static final java.lang.String MPEG4_480_VIDEO
```

Compresses video using the MPEG-4 codec using medium bit-rate settings with dimensions no larger than 720x480. This is appropriate for delivery to medium and high-bandwidth and medium- and high-capacity destinations. Only available in 32-bit.

See Also:
Constant Field Values

LOSSLESS_ALAC_AUDIO

```
public static final java.lang.String LOSSLESS_ALAC_AUDIO
```

Compresses audio using the Apple Lossless codec. This is appropriate for an intermediate format for media that requires further processing.

See Also:
Constant Field Values

HQ_AAC_AUDIO

```
public static final java.lang.String HQ_AAC_AUDIO
```

Compresses audio using the AAC codec at 64 kbps per channel. This is appropriate for delivery of high-quality music and other audio.

See Also:
Constant Field Values

VQ_AAC_AUDIO

```
public static final java.lang.String VQ_AAC_AUDIO
```

Compresses audio using the AAC codec at 32 kbps per channel. This is appropriate for delivery of voice recordings.

See Also:
Constant Field Values

Constructor Detail

CompressionOptions

```
public CompressionOptions(java.lang.String identifier)
                  throws CompressionOptions.InvalidCompressionException
```

Initializes a newly created CompressionOptions object configured for the given identifier.

Parameters:
`identifier` - the identifier for the compression options object
Throws:
`CompressionOptions.InvalidCompressionException`

Method Detail

toString

```
public java.lang.String toString()
```

Overrides:
`toString` in class `java.lang.Object`

---

equals

```
public boolean equals(java.lang.Object k)
```

Overrides:
`equals` in class `java.lang.Object`

---

hashCode

```
public int hashCode()
```

Overrides:
`hashCode` in class `java.lang.Object`

getCompressionOptionsAvailableForMediaType

```
public static java.util.List<CompressionOptions> getCompressionOptionsAvailableForMediaType(MediaType mediaType)
```

Returns the list of all possible and available compression options for the given media type on the userâs system.

Returns:

returns a `List` of CompressionOptions instances of all possible and available compression options for the given media type on the userâs system

---

getMediaType

```
public MediaType getMediaType()
```

Returns the media type on which the receiverâs compression options should be used.

Returns:

the media type, such as Mediatype.VIDEO, Mediatype.SOUND, or Mediatype.MUXED

---

getLocalizedDisplayName

```
public java.lang.String getLocalizedDisplayName()
```

Returns a short localized name describing the receiverâs compression options.

Returns:

a localized string appropriate for display in the user interface (in a list of compression options, for example)

---

getLocalizedCompressionOptionsSummary

```
public java.lang.String getLocalizedCompressionOptionsSummary()
```

Returns a localized summary of the receiverâs compression options.

Returns:

a localized string summarizing the receiver’s compression options

---

it.unitn.science.JCQLibrary

Class JCQFactory

```
java.lang.Object
  extended by  it.unitn.science.JCQLibrary.JCQFactory
```

```
public class JCQFactoryextends java.lang.Object
```

The JCQFactory class defines the simple way for capturing and previewing from a video and a microphone device. You use the singleton instance of this class to call the basic methods, you get the singleton instance by a call to `getInstance()` method.

Applications can add a microphone, video or muxed device to the singleton instance of this class with `setCameraDevice(CaptureDevice)` and `setMicrophoneDevice(CaptureDevice)` as inputs. Then application have to specify the path for the output file where the recording sample buffer will be writed with `setOutputFilePath(String)`, you can also set a preview `CaptureView` as output of the session with `setCaptureView(CaptureView)`. Then you start the capture session with `openSession()` and remember to close at the end with `closeSession()`. The you can control the recording with: `startRecording()`, `stopRecording()`, `pauseRecording()`, `resumeRecording()`, and start or stop previewing with `startPreviewing()` and `stopPreviewing()`.

Since:
1.5

Method Summary	
void	**closeSession()** Attempts to stop running the associated session of the factory.
CompressionOptions	**getAudioCompressionOptions()** Returns the setted audio compression options for the output file.
CaptureDevice	**getCameraDevice()** Returns the current setted camera device.
CaptureMovieFileOutput	**getCaptureMovieFileOutput()** Return the capture moview file output to which the session will recording.
CaptureSession	**getCaptureSession()** Returns the capture session of the singleton instace of the factory.
CaptureView	**getCaptureView()** Returns the capture view for which the session will output the preview.
static JCQFactory	**getInstance()** Returns the singleton JCQFactory instance object.
CaptureDevice	**getMicrophoneDevice()** Returns the current setted microphone device.
java.lang.String	**getOutputFilePath()** Returns the file path of the current output file.

CompressionOptions	**getVideoCompressionOptions**() Returns the setted video compression options for the output file.
boolean	**isOpenCapturedFile**() Return if when the file written to by the receiver is closed have to be opened.
boolean	**isPreviewing**() Return the current state of previewing.
boolean	**isRecording**() Return the current state of recording.
void	**openSession**() Attempts to start running the associated session of the factory.
void	**pauseRecording**() Attempts to pause recording.
void	**resumeRecording**() Attempts to resume recording.
void	**setAudioCompressionOptions**(CompressionOptions compression) Sets the audio compression of the file written to by the receiver, specifying the desired audio compression.
void	**setCameraDevice**(CaptureDevice device) Sets the camera device.
void	**setCaptureView**(CaptureView view) Sets the capture view for the previewing
void	**setMicrophoneDevice**(CaptureDevice device) Sets the microphone device.
void	**setOpenCapturedFile**(boolean flag) Sets if when the file written to by the receiver is closed have to be opened.
void	**setOutputFilePath**(java.lang.String outputFilePath) Sets the file path for the output file where the sample buffer will be writed.
void	**setVideoCompressionOptions**(CompressionOptions compression) Sets the video compression of the file written to by the receiver, specifying the desired video compression.
void	**startPreviewing**() Attempts to start previewing.
void	**startRecording**() Attempts to start recording.
void	**stopPreviewing**() Attempts to stop previewing.
void	**stopRecording**() Attempts to stop recording.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

getInstance

```
public static JCQFactory getInstance()
```

Returns the singleton JCQFactory instance object.

Returns:
returns the singleton factory

---

getCameraDevice

```
public CaptureDevice getCameraDevice()
```

Returns the current setted camera device.

Returns:
returns an instance of `CaptureDevice` with media type MediaType.VIDEO or MediaType.MUXED, that is currently the setted camera device in the factory or `null` otherwise

---

getMicrophoneDevice

```
public CaptureDevice getMicrophoneDevice()
```

Returns the current setted microphone device.

Returns:
returns an instance of `CaptureDevice` with media type MediaType.SOUND, that is currently the setted microphone device in the factory or `null` otherwise

---

getVideoCompressionOptions

```
public CompressionOptions getVideoCompressionOptions()
```

Returns the setted video compression options for the output file.

Returns:
returns an instance of `CompressionOptions`

---

getAudioCompressionOptions

```
public CompressionOptions getAudioCompressionOptions()
```

Returns the setted audio compression options for the output file.

Returns:
Returns an instance of `CompressionOptions`

---

getOutputFilePath

```
public java.lang.String getOutputFilePath()
```

Returns the file path of the current output file.

Returns:
returns the string path of the file

---

getCaptureSession

```
public CaptureSession getCaptureSession()
```

Returns the capture session of the singleton instace of the factory.

Returns:
returns an instance of `CaptureSession`

---

getCaptureView

```
public CaptureView getCaptureView()
```

Returns the capture view for which the session will output the preview.

Returns:
returns an instance of `CaptureView`

---

getCaptureMovieFileOutput

```
public CaptureMovieFileOutput getCaptureMovieFileOutput()
```

Return the capture moview file output to which the session will recording.

Returns:
returns an instance of `CaptureMovieFileOutput`

---

isPreviewing

```
public boolean isPreviewing()
```

Return the current state of previewing.

Returns:
`true` if the factory is previewing, `false` otherwise

---

isRecording

```
public boolean isRecording()
```

Return the current state of recording.

Returns:
`true` if the factory is recording, `false` otherwise

---

isOpenCapturedFile

```
public boolean isOpenCapturedFile()
```

Return if when the file written to by the receiver is closed have to be opened. The method tell if when the output file is closed by a `stopRecording()` have to be opened with the default player application (eg. QuickTime).

Returns:
`true` if the file will be opened, `false` otherwise

---

setCameraDevice

```
public void setCameraDevice(CaptureDevice device)
```

Sets the camera device. If a camera device was precedent setted, this method first close it and remove from the session. If the passed device is not a camera one this method does nothing. If the camera is muxed, like DV Camera, it will also be used as sound input.

Parameters:
`device` - the desired input capture device

---

setMicrophoneDevice

```
public void setMicrophoneDevice(CaptureDevice device)
```

Sets the microphone device. If a microphone device was precedent setted, this method first close it and remove from the session. If the passed device is not a microphone one this method does nothing.

Parameters:
`device` - the desired input capture device

---

setVideoCompressionOptions

```
public void setVideoCompressionOptions(CompressionOptions compression)
```

Sets the video compression of the file written to by the receiver, specifying the desired video compression. The method sets the video compression of the file to which the receiver is currently writing output media. If the passed compression is not a kind of video one it does nothing.

Parameters:
`compression` - object containing the desired video compression of the output file.
See Also:
`CompressionOptions`

---

setAudioCompressionOptions

```
public void setAudioCompressionOptions(CompressionOptions compression)
```

Sets the audio compression of the file written to by the receiver, specifying the desired audio compression. The method sets the audio compression of the file to which the receiver is currently writing output media. If the passed compression is not a kind of sound one it does nothing.

Parameters:
`compression` - object containing the desired audio compression of the output file.
See Also:
`CompressionOptions`

---

setOutputFilePath

```
public void setOutputFilePath(java.lang.String outputFilePath)
```

Sets the file path for the output file where the sample buffer will be writed. If the file exist it will be overwrited, otherwise it will be created.

Parameters:
`outputFilePath` - the string path of the output file

---

setCaptureView

```
public void setCaptureView(CaptureView view)
```

Sets the capture view for the previewing

Parameters:
`view` - the AWT component instance of `CaptureView`

---

setOpenCapturedFile

`public void` **`setOpenCapturedFile`**`(boolean flag)`

Sets if when the file written to by the receiver is closed have to be opened. The method sets if when the file written to by the receiver is closed by a `stopRecording()` have to be opened with the default player application (eg. QuickTime).

Parameters:

`flag` - if `true` the file will be opened if `false` not

---

openSession

`public void` **`openSession`**`()`

Attempts to start running the associated session of the factory. The inputs will be send to the outputs.

---

closeSession

`public void` **`closeSession`**`()`

Attempts to stop running the associated session of the factory. The inputs will be remove and close and the output will be close.

---

startPreviewing

`public void` **`startPreviewing`**`()`

Attempts to start previewing.

---

stopPreviewing

`public void` **`stopPreviewing`**`()`

Attempts to stop previewing.

---

startRecording

`public void` **`startRecording`**`()`

Attempts to start recording.

---

pauseRecording

```
public void pauseRecording()
```

Attempts to pause recording. The recording can be resume with `resumeRecording()`.

---

resumeRecording

```
public void resumeRecording()
```

Attempts to resume recording.

---

stopRecording

```
public void stopRecording()
```

Attempts to stop recording. The ouput file will be closed and open if precedent setted to open with `setOpenCapturedFile(boolean)`.

---

it.unitn.science.JCQLibrary

Enum MediaType

```
java.lang.Object
  extended by  java.lang.Enum<MediaType>
      extended by  it.unitn.science.JCQLibrary.MediaType
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable<MediaType>

```
public enum MediaTypeextends java.lang.Enum<MediaType>
```

This enum represents a particular type of media.

Since:

1.5

Enum Constant Summary

MUXED	Multiplexed media that may contain audio, video, and other data in a single stream.
SOUND	Media that only contains audio samples.
VIDEO	Media that only contains video frames.

Method Summary

static MediaType	**valueOf**(java.lang.String name) Returns the enum constant of this type with the specified name.
static MediaType[]	**values**() Returns an array containing the constants of this enum type, in the order they are declared.

Methods inherited from class java.lang.Enum
clone, compareTo, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

Methods inherited from class java.lang.Object
getClass, notify, notifyAll, wait, wait, wait

Enum Constant Detail

VIDEO

```
public static final MediaType VIDEO
```

Media that only contains video frames. iSight cameras (external and built-in); USB and FireWire webcams.

MUXED

```
public static final MediaType MUXED
```

Multiplexed media that may contain audio, video, and other data in a single stream. DV cameras

---

SOUND

```
public static final MediaType SOUND
```

Media that only contains audio samples. Built-in microphones and line-in jacks; the microphone built-in to the external iSight; USB microphones and headsets; any other device supported by Core Audio.

Method Detail

values

```
public static MediaType[] values()
```

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (MediaType c : MediaType.values())
    System.out.println(c);
```

Returns:

an array containing the constants of this enum type, in the order they are declared

---

valueOf

```
public static MediaType valueOf(java.lang.String name)
```

Returns the enum constant of this type with the specified name. The string must match *exactly* an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.)

Parameters:

`name` - the name of the enum constant to be returned.

Returns:

the enum constant with the specified name

Throws:

`java.lang.IllegalArgumentException` - if this enum type has no constant with the specified name

`java.lang.NullPointerException` - if the argument is null

Bibliografia

[1] Chris Adamson. *QuickTime for Java: A Developer's Notebook.* O'Reilly Media, 2005.

Sitografia

[1] Apple's 1984 Commercial. http://www.uriahcarpenter.info/1984.html.

[2] Apple Mailing List. QTJava will be depreciated next year. http://lists.apple.com/archives/quicktime-java/2008/Jun/msg00018.html.

[3] Apple Inc. QuickTime for Java. http://developer.apple.com/quicktime/qtjava/.

[4] Apple Inc. Java Native Foundation Functions Reference. http://developer.apple.com/library/mac/#documentation/CrossPlatform/Reference/JavaNativeFoundation_Functions/Reference/reference.html.

[5] Apple Inc. MPEG-4 The container for digital media. http://web.archive.org/web/20070111065647/http://www.apple.com/quicktime/technologies/mpeg4/. Archiviato dall'originale http://www.apple.com/quicktime/technologies/mpeg4/ non più accessibile.

[6] Apple Inc. QTKit Framework Reference. http://developer.apple.com/library/mac/#documentation/QuickTime/Reference/QTCocoaObjCKit/_index.html.

[7] Apple Inc. Quicktime. http://www.apple.com/quicktime.

[8] Apple Inc. QuickTime Javadoc. http://developer.apple.com/quicktime/qtjava/javadocs.html.

[9] Apple Inc. Technical note TN2147: JNI development on Mac OS X. http://developer.apple.com/library/mac/#technotes/tn2005/tn2147.html.

[10] Matteo Matassoni. Java Capture with QuickTime Javadoc. http://tinyurl.com/JCQ-Javadoc.

[11] Marco Ronchetti. Lectures On DEmand. http://latemar.science.unitn.it/segue/index.php?&action=site&site=LODE.

[12] Marco Ronchetti. Building with qtjava a video recorder that allows previewing while recording. http://marcoronchetti.blogspot.com/2008/05/bulding-video-recorder-that-allows.html.

[13] Sun Microsystems, Inc. The Java Native Interface: Programmer's Guide and Specification. http://java.sun.com/docs/books/jni/html/preface.html.

Lista acronimi

API Application Programming Interface 1

AWT Abstract Window Toolkit 31

IDE Integrated Development Environment 8

JNI Java Native Interface 1

JVM Java Virtual Machine 22

JCQ Java Capture with QuickTime 1

JMF Java Media Framework 44

LODE Lectures On DEmand 2

OOP Object-oriented programming 7

QTKit QuickTime Kit Framework 1

QTJava QuickTime for Java 1

WWDC Worldwide Developer Conference 6

UI User Interface 36