**ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ**

**Σχολή Τεχνολογικών Εφαρμογών**
**Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων**

# Πτυχιακή Εργασία

# "Συρραφή βίντεο σε πραγματικό χρόνο"

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: **ΚΑΜΠΙΤΑΚΗΣ ΝΕΚΤΑΡΙΟΣ**

ΕΙΣΗΓΗΤΗΣ: **ΜΑΛΑΜΟΣ ΑΘΑΝΑΣΙΟΣ**

ΗΡΑΚΛΕΙΟ 2008

## Ευχαριστίες

Με την ολοκλήρωση της πτυχιακής μου εργασίας, θα ήθελα να ευχαριστήσω όλους όσοι βοήθησαν στην περάτωση αυτής. Ευχαριστώ θερμά όλους εκείνους που ήταν δίπλα μου σε όλη αυτή την προσπάθεια, παρέχοντας μου απεριόριστη ψυχολογική υποστήριξη και κατανόηση.

Ηράκλειο, Μάρτιος 2008

# *Περιεχόμενα*

# Περιεχόμενα εικόνων



# Περιεχόμενα σχημάτων



# Περιεχόμενα πινάκων

# 1. Εισαγωγή

## 1.1. Γενική περιγραφή

Η πτυχιακή εργασία που περιγράφεται στο παρακάτω κείμενο ασχολείται με τη δημιουργία μίας εφαρμογής η οποία θα κάνει συρραφή βίντεο ή συρραφή εικόνων. Η εφαρμογή αυτή μέσω της πλατφόρμας εργασίας που δημιουργεί, παρέχει την δυνατότητα στο χρήστη να προχωρήσει στη συρραφή των αρχείων. Το παραγόμενο αρχείο της εφαρμογής είναι ένα βίντεο το οποίο και αποθηκεύεται στο δίσκο. Η εφαρμογή που υλοποιήθηκε στηρίζεται στη γλώσσα προγραμματισμού Java και στα API (Application Programming Interface) Java Media Framework (JMF) και JFC (Swing) της Java. Σύμφωνα με το βασικό σενάριο γίνεται εισαγωγή των βίντεο ή των εικόνων στην πλατφόρμα εργασίας. Η εισαγωγή των αρχείων (εικόνες ή βίντεο) μπορεί να γίνει επιλέγοντας είτε ένα αρχείο ξεχωριστά είτε ένα ολόκληρο φάκελο με αρχεία. Στη συνέχεια αφού επιλεχθεί η σειρά με την οποία τα βίντεο ή οι εικόνες θα ενωθούν, γίνεται η συρραφή τους. Το παραγόμενο βίντεο αποθηκεύεται στο δίσκο από όπου στη συνέχεια μπορεί να αναπαραχθεί.

## 1.2. Στόχος

Ο στόχος της εργασίας είναι η υλοποίηση μίας εφαρμογής η οποία απλά και γρήγορα θα κάνει συρραφή βίντεο ή συρραφή εικόνων σε πραγματικό χρόνο. Στην εφαρμογή τα βίντεο ενώνονται μόνο με βίντεο, όπως και οι εικόνες ενώνονται μόνο με εικόνες. Η συρραφή βίντεο ή η συρραφή εικόνων που είναι κωδικοποιημένα με διαφορετικό Format ή με διαφορετικές διαστάσεις, είναι επίσης ένας στόχος που υλοποιήθηκε. Για να γίνει λοιπόν δυνατή η συρραφή των βίντεο ή των εικόνων που είτε είναι κωδικοποιημένα σε διαφορετικό Format είτε έχουν διαφορετικές διαστάσεις, πρώτα γίνεται επανακωδικοποίηση των αρχείων με ένα κοινό Format και με τις ίδιες διαστάσεις αντίστοιχα και στην συνέχεια ακολουθεί η συρραφή τους.

## 1.3. Εργαλεία υλοποίησης

Για την υλοποίηση της εφαρμογής χρησιμοποιήθηκε η γλώσσα προγραμματισμού Java και τα πακέτα βιβλιοθηκών που αυτή προσφέρει. Η Java σε

συνδυασμό με το API Java Media Framework (JMF) αποτελεί ένα πολύ ισχυρό εργαλείο για την διαχείριση και επεξεργασία αρχείων πολυμέσων (βίντεο, εικόνων, κ.α.). Έτσι λοιπόν η χρήση των βιβλιοθηκών του JMF ήταν καθοριστική για την υλοποίηση της εφαρμογής. Επίσης για την υλοποίηση του GUI (Graphical User Interface) της εφαρμογής χρησιμοποιήθηκε κυρίως το Swing το οποίο είναι μέρος του API Java Foundation Classes (JFC) της Java. Κάθε API της Java είναι μία μεγάλη συλλογή από έτοιμα προγράμματα της Java τα οποία παρέχουν χρήσιμες δυνατότητες σε ένα υπό κατασκευή πρόγραμμα.

### *1.3.1. Java*

Η γλώσσα προγραμματισμού Java λόγω των δυνατοτήτων που προσφέρει στην διαχείριση αρχείων βίντεο, εικόνας κ.α.. αποτελεί ίσως την ιδανικότερη επιλογή για την ανάπτυξη εφαρμογών που χρησιμοποιούνται για την παρουσίαση και επεξεργασία αρχείων πολυμέσων. Η Java αποτελεί την κυρίαρχη και βασική ιδέα στην εφαρμογή. Η εφαρμογή στηρίζεται σε κώδικα που γράφηκε σε Java. Έτσι λοιπόν τα πάντα στην εφαρμογή αυτή λειτουργούν με την χρήση της Java και των βιβλιοθηκών που αυτή προσφέρει.

### *1.3.2. Java Media Framework*

Το API Java Media Framework είναι μία βιβλιοθήκη πακέτων της Java η όποια προσφέρει πολλές δυνατότητες στην διαχείριση αρχείων πολυμέσων και υποστηρίζει σύνθετη επεξεργασία εικόνας, ήχου και βίντεο, αναγνωρίζοντας τους περισσότερους τύπους αρχείων που κυκλοφορούν. Το JMF χρησιμοποιείται για να μπορέσει κάποιος να εισαγάγει αρχεία πολυμέσων σε μία εφαρμογή που έχει υλοποιηθεί με Java και να τα διαχειριστεί ανάλογα. Έχει σχεδιαστεί για να προσφέρει, σε έναν προγραμματιστή, όλα όσα χρειάζεται για να υλοποιήσει μία εφαρμογή στην οποία θα έχει επιπρόσθετο έλεγχο στην αναπαραγωγή και επεξεργασία αρχείων πολυμέσων. Η αρχιτεκτονική του JMF είναι απλή και αποτελείται από τρία βασικά στάδια. Στο πρώτο στάδιο γίνεται η απόκτηση ενός αρχείου πολυμέσων, στο δεύτερο στάδιο η επεξεργασία του αρχείου και τέλος στο τρίτο στάδιο το αρχείο είτε αποθηκεύεται είτε παρουσιάζεται (αναπαράγαγεται). Το JMF λοιπόν, σχηματοποιεί και επεκτείνει τις δυνατότητες για την διαχείριση

πολυμέσων στην Java, και δίνει στους προγραμματιστές ένα ισχυρό κουτί εργαλείων για την ανάπτυξη εφαρμογών διαχείρισης αρχείων πολυμέσων.

Η βιβλιοθήκη πακέτων JMF αποτελεί τον ακρογωνιαίο λίθο για την υλοποίηση της εφαρμογής. Η έκδοση του JMF (2.1.1e) που χρησιμοποιείται για αυτή την εφαρμογή αποτελείται από 209 κλάσεις και είναι χωρισμένη σε 11 πακέτα εκ των οποίων τα ακόλουθα χρησιμοποιούνται στην εφαρμογή:

- javax.media: Το κύριο πακέτο που βρίσκεται στο πρώτο επίπεδο και περιλαμβάνει τις περισσότερες κλάσεις αλλά και μερικές από τις πιο σημαντικές, όπως τις: Manager, Processor, DataSink και Player.
- javax.media.control: Το πακέτο αυτό αποτελείται από 18 κλάσεις που καθορίζουν τους διαφορετικούς τύπους των Controls (π.χ.: TrackControl και FormatControl).
- javax.media.format: Ένα σημαντικό πακέτο 10 κλάσεων που καθορίζουν τα διαφορετικά Format τα οποία το JMF είναι ικανό να διαχειριστεί (π.χ.: AudioFormat και H263Format).
- javax.media.protocol: Ένα σημαντικό πακέτο 26 κλάσεων οι οποίες παρέχουν την υποστήριξη για την επικοινωνία με το DataSource. Το DataSource αναπαριστά αρχεία πολυμέσων στο JMF (π.χ.: FileTypeDescriptor και PullBufferDataSource).
- javax.media.util: Ένα πακέτο δύο πολύ χρήσιμων κλάσεων: BufferToImage και ImageToBuffer που χρησιμοποιούνται για μετατροπή μεταξύ JMF Buffers και AWT εικόνων.
- javax.media.datasink: Ένα πακέτο τεσσάρων κλάσεων που χρησιμοποιούνται για εξαγωγή γεγονότων που προκύπτουν κατά την λειτουργία του DataSink. Το DataSink χρησιμοποιείται για την αποθήκευση αρχείων (π.χ.: DataSinkErrorEvent και DataSinkListener).

### *1.3.3. Java Foundation Classes (JFC)*

Η βιβλιοθήκη πακέτων Swing (το Swing αποτελεί μέρος του JFC) περιλαμβάνει πάρα πολλά συστατικά τα οποία χρησιμοποιούνται για την δημιουργία του γραφικού περιβάλλοντος (GUI) μιας εφαρμογής και ουσιαστικά είναι μία

ουσιώδης επέκταση της βιβλιοθήκης πακέτων AWT. Η γκάμα των συστατικών του Swing περιλαμβάνει κουμπιά, πλαίσια ελέγχου, ετικέτες, περιγράμματα, μενού, λεζάντες και πολλά άλλα. Τα συστατικά του Swing έχουν όνομα που ξεκινά με το γράμμα J, ακολουθείται από ένα ακόμα κεφαλαίο γράμμα και στη συνέχεια υπάρχει το υπόλοιπο όνομα που γράφεται με μικρά γράμματα. Πολλά από τα συστατικά του Swing χρησιμοποιήθηκαν για την υλοποίηση του GUI της εφαρμογής που παρουσιάζεται.

## *1.4. Video - Audio Formats*

Στον παρακάτω πίνακα παρουσιάζονται όλα τα Video και Audio Formats τα οποία το JMF υποστηρίζει. Έτσι στην εφαρμογή είναι δυνατό αρχεία βίντεο ή ήχου ή ο συνδυασμός και των δύο, τα οποία είναι κωδικοποιημένα με τα Format που φαίνονται στον παρακάτω πίνακα, να αναπαραχθούν ή να διαχειριστούν και να δώσουν το κατάλληλο αποτέλεσμα. Παρόλο το ευρύ φάσμα των Video και Audio Format που το JMF υποστηρίζει υπάρχει ένας περιορισμός. Το αρχείο πολυμέσων που θα εξαχθεί από την εφαρμογή μπορεί να είναι τύπου QuickTime ή AVI. Αυτό συμβαίνει επειδή ο Multiplexer (το JMF τον παρέχει μέσω της κλάσης Processor) που χρησιμοποιείται για την εξαγωγή του αρχείου βίντεο υποστηρίζει μόνο τα αρχεία τύπου (Content Type) QuickTime και AVI. Έτσι λοιπόν στην εφαρμογή το αρχείο εξόδου δεν θα μπορεί να είναι για παράδειγμα τύπου MPEG, παρά μόνο τύπου QuickTime και AVI. Όσον αφορά τα αρχεία των Windows Media, αυτά δεν αναγνωρίζονται από το JMF. Αυτό σημαίνει ότι τα αρχεία αυτά (wmv, asf) δεν δύναται να χειριστούν με την χρήση του JMF, οπότε ούτε και με την εφαρμογή.

.

| **Video** | | **Audio** | |
|---|---|---|---|
| **Format** | **Content Type** | **Format** | **Content Type** |
| Cinepak | QuickTime, AVI | PCM | QuickTime, AVI, WAV |
| MPEG-1 | MPEG | Mu-Law | QuickTime, AVI, WAV |
| H.261 | AVI | ADPCM(DVI,IMA4) | QuickTime, AVI, WAV |
| H.263 | QuickTime, AVI | MPEG-1 | MPEG |
| JPEG | QuickTime, AVI | MPEG Layer3 | MPEG |
| Indeo | QuickTime, AVI | GSM | WAV |

**Πίνακας 1**

# 2. Η εφαρμογή

## 2.1. Γενική παρουσίαση της εφαρμογής

Η γενική ιδέα της εφαρμογής βασίζεται στην συρραφή βίντεο και την συρραφή εικόνων. Για να είναι η εφαρμογή εύκολη στη χρήση δημιουργήθηκε μία πλατφόρμα εργασίας, στην οποία μπορούν να εκτελεστούν όλες οι λειτουργίες της. Παρακάτω παρουσιάζονται μερικές από τις λειτουργίες της εφαρμογής.

Από το μενού της εφαρμογής, ο χρηστής, μπορεί να επιλέξει να εμφανίσει μία εικόνα ή να αναπαράγει ένα βίντεο. Η αναπαραγωγή του βίντεο γίνεται με την χρήση ενός Media Player ο οποίος δίνει στο χρήστη τις βασικές δυνατότητες ενός Player όπως: Play, Pause, έλεγχος της ροής αναπαραγωγής, ρύθμιση της έντασης του ήχου κ.α.. Μία άλλη βασική λειτουργία είναι η επιλογή των αρχείων τα οποία ο χρήστης επιθυμεί να ενώσει. Η επιλογή των αρχείων γίνεται είτε μαζικά, επιλέγοντας ένα ολόκληρο φάκελο, εισάγοντας έτσι όλα τα αρχεία που βρίσκονται μέσα στο φάκελο, είτε ένα αρχείο κάθε φορά. Αφού επιλέξει τα αρχεία, ο χρήστης μπορεί να αλλάξει τη σειρά τους ή να διαγράψει ένα από αυτά. Στην περίπτωση όπου τα αρχεία προς συρραφή είναι εικόνες ο χρήστης μπορεί να επιλέξει το FrameRate που θα έχουν οι εικόνες στο βίντεο ή τις διαστάσεις (width, height) που επιθυμεί να έχει το παραγόμενο βίντεο. Τέλος αφού ο χρήστης επιλέξει που θα αποθηκευτεί το παραγόμενο βίντεο και το όνομα αυτού, μπορεί να προχωρήσει στην συρραφή των αρχείων.

Κατά την επιλογή του ονόματος του παραγόμενου αρχείου ο χρήστης επιλέγει τον τύπο εξόδου του αρχείου ανάλογα με την κατάληξη (.avi) που θα δώσει σε αυτό. Οι επιλογές του χρήστη είναι δύο (.avi ή .mov) για το λόγο που εξηγήθηκε στην προηγούμενη ενότητα. Με τον τρόπο αυτό δίνεται η δυνατότητα στην εφαρμογή να αντιληφθεί τον τύπο για το παραγόμενο βίντεο. Αφού γίνει η συρραφή των αρχείων (σωστά και χωρίς προβλήματα), μπορεί να γίνει αναπαραγωγή του βίντεο που έδωσε η εφαρμογή.

Αν κατά την διάρκεια της συρραφής δημιουργηθεί κάποιο πρόβλημα, τότε η διαδικασία σταματάει άμεσα και εμφανίζεται ένα μήνυμα με περιεχόμενο την αιτία για την οποία η διαδικασία τερματίστηκε.

## 2.2. Σενάριο συρραφής εικόνων

Στην παράγραφο αυτή γίνεται συνοπτική περιγραφή της διαδικασίας που ακολουθείται στην περίπτωση που ο χρήστης αποφασίσει να κάνει συρραφή εικόνων. Αφού λοιπόν ο χρήστης επιλέξει τις εικόνες που επιθυμεί να ενώσει αρχίζει η διαδικασία της συρραφής των εικόνων σε βίντεο. Αρχικά από την κατάληξη του ονόματος που έχει δώσει ο χρήστης για το παραγόμενο βίντεο, η εφαρμογή εντοπίζει τον τύπο (mov, avi) του παραγόμενου βίντεο. Στη συνέχεια αν ο χρήστης δεν έχει δώσει τις διαστάσεις, τις οποίες επιθυμεί να έχει το παραγώμενο βίντεο και οι εικόνες έχουν διαφορετικές διαστάσεις, γίνεται διαμόρφωση των διαστάσεων κάθε εικόνας σύμφωνα με τις διαστάσεις της πρώτης εικόνας. Αν όμως ο χρήστης έχει δώσει τις διαστάσεις που επιθυμεί να έχει το βίντεο τότε γίνεται διαμόρφωση των διαστάσεων κάθε εικόνας σύμφωνα με τις διαστάσεις που έχει δώσει ο χρήστης. Τέλος η εφαρμογή μετατρέπει κάθε εικόνα σε Buffer (προσωρινή αποθήκη για ρεύμα δεδομένων), δίνει στο Buffer την διάρκεια (FrameRate) που θα έχει κάθε εικόνα στο βίντεο και τις υπόλοιπες πληροφορίες για το Format και στέλνει το Buffer για αποθήκευση στο δίσκο.

## 2.3. Σενάριο συρραφής βίντεο

Στην παράγραφο αυτή γίνεται συνοπτική περιγραφή της διαδικασίας που ακολουθείται στην περίπτωση που ο χρήστης αποφασίσει να κάνει συρραφή βίντεο. Αφού λοιπόν ο χρήστης επιλέξει τα βίντεο που επιθυμεί να ενώσει αρχίζει η διαδικασία της συρραφής όλων των βίντεο σε ένα.

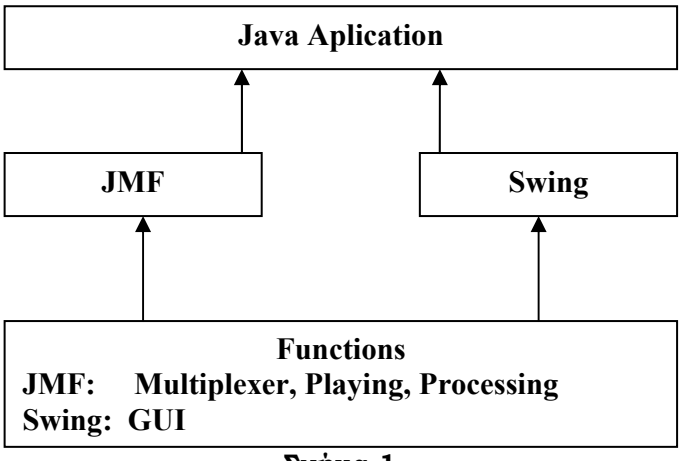
Αρχικά από την κατάληξη του ονόματος που έχει δώσει ο χρήστης για το παραγόμενο βίντεο η εφαρμογή εντοπίζει τον τύπο (avi, mov) του παραγόμενου βίντεο. Στη συνέχεια για κάθε αρχείο γίνονται οι ακόλουθες ενέργειες: αν το αρχείο περιέχει και βίντεο και ήχο (2 Tracks) γίνεται διαχωρισμός του βίντεο και του ήχου με την βοήθεια της κλάσης TrackControl. Η εφαρμογή παίρνει τα Tracks(βίντεο, ήχος) κάθε αρχείου και εξετάζει αν αυτά (τα Tracks βίντεο όλων των αρχείων και τα Tracks ήχου όλων των αρχείων αντίστοιχα) χρειάζονται επανακωδικοποίηση ώστε να είναι δυνατό να γίνει η συρραφή τους. Στην περίπτωση που χρειάζεται επανακωδικοποίηση, τότε εντοπίζεται ένα Format για το Track βίντεο και ένα Format για το Track ήχου τα οποία υποστηρίζονται από τα Tracks βίντεο και ήχου όλων των

αρχείων αντίστοιχα. Τα Format αυτά των Tracks βίντεο και ήχου εφαρμόζονται στα Tracks βίντεο και ήχου όλων των αρχείων αντίστοιχα. Επίσης εξετάζονται οι διαστάσεις των βίντεο. Αν είναι διαφορετικές τότε εφαρμόζονται οι διαστάσεις του πρώτου βίντεο σε όλα τα υπόλοιπα βίντεο. Αν η εφαρμογή δεν μπορέσει να εντοπίσει Format τα οποία υποστηρίζονται από όλα τα Tracks των αρχείων τότε η διαδικασία τερματίζεται. Τέλος όλα τα αρχεία μετατρέπονται σε Buffer και αφού δοθούν οι πληροφορίες για την διάρκεια όλων των Tracks έτσι ώστε αυτά να συγχρονιστούν σωστά, το Buffer αποθηκεύεται στο δίσκο.

## 2.4. Αρχιτεκτονικός σχεδιασμός εφαρμογής

### 2.4.1. Δομή εφαρμογής

Όπως αναφέρθηκε προηγουμένως, η εφαρμογή υλοποιήθηκε αποκλειστικά με την χρήση της γλώσσας προγραμματισμού Java. Με βάση λοιπόν την γλώσσα προγραμματισμού και τα τις βιβλιοθήκες πακέτων που χρησιμοποιήθηκαν, η δομή της εφαρμογής χωρίζεται σε τρία βασικά επίπεδα. Στο πρώτο επίπεδο βρίσκεται η γλώσσα προγραμματισμού Java. Στο δεύτερο επίπεδο έχουμε κυρίως τις βιβλιοθήκες πακέτων JMF και Swing (ελάχιστα επίσης χρησιμοποιήθηκε η βιβλιοθήκη πακέτων AWT), οι οποίες χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής. Στο τρίτο επίπεδο έχουμε τις λειτουργίες που υλοποιήθηκαν με βάση τις κλάσεις από τις βιβλιοθήκες πακέτων που βρίσκονται στο προηγούμενο επίπεδο. Το σχήμα που ακολουθεί αποδίδει την δομή της εφαρμογής.

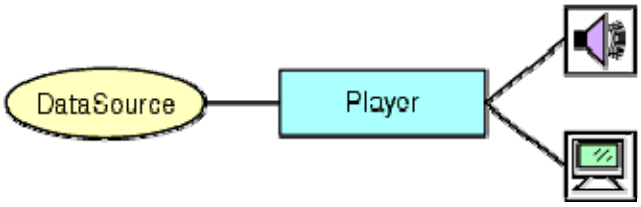



**Σχήμα 1**

Στο παραπάνω σχήμα γίνεται αντιληπτό πως για την υλοποίηση της εφαρμογής χρησιμοποιήθηκαν η Java και οι βιβλιοθήκες της JMF και Swing. Για να γίνει περισσότερο κατανοητή η δομή της εφαρμογής θα χρησιμοποιηθεί ως παράδειγμα η αναπαραγωγή ενός βίντεο. Για την αναπαραγωγή ενός βίντεο λοιπόν, η Java χρησιμοποιεί τις βιβλιοθήκες πακέτων JMF και Swing για να πάρει τις κλάσεις από αυτές, με την βοήθεια των οποίων θα εισάγει το αρχείο στην εφαρμογή, θα το χειριστεί κατάλληλα και τέλος θα δημιουργήσει το GUI για να γίνει δυνατή η προβολή του βίντεο στο χρήστη.

### *2.4.2. Σχεδιασμός της διαδικασίας αναπαραγωγής βίντεο*

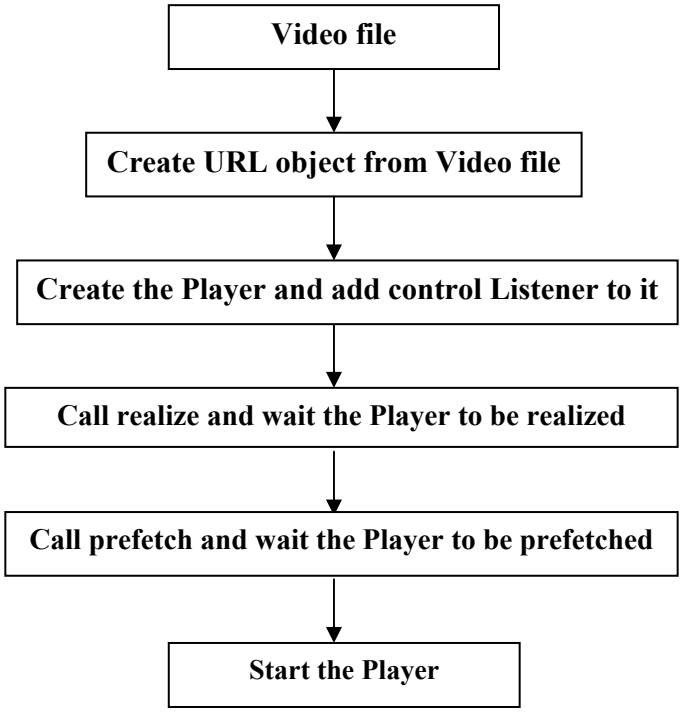
Με την χρήση της κλάσης Player της JMF αναπαράγεται ένα βίντεο σε μία Java εφαρμογή. Η λειτουργία που κάνει ο Player είναι απλή όπως φαίνεται στο παρακάτω σχήμα.




**Σχήμα 2**

Ο Player λοιπόν παίρνει το βίντεο, το αποκωδικοποιεί για να μπορέσει να το αναπαράγει, χωρίς όμως να έχει την δυνατότητα να επέμβει (π.χ. να αλλάξει το Format) σε αυτό και το παρουσιάζει στον χρήστη.

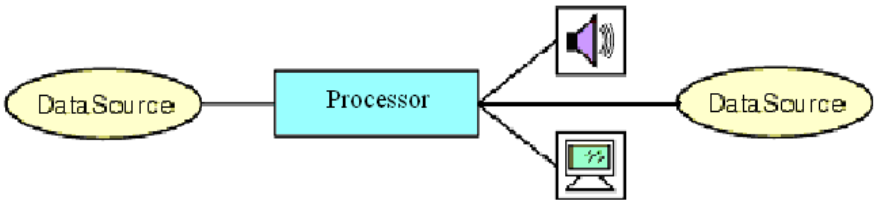
Όσον αφορά την διαδικασία αναπαραγωγής ενός βίντεο, αφού ο χρήστης επιλέξει το αρχείο που επιθυμεί να παρακολουθήσει, δημιουργείται ένα αντικείμενο τύπου URL με παράμετρο τη διαδρομή στο δίσκο και το όνομα του αρχείου. Στην συνέχεια δημιουργείται ο Player (αντικείμενο της κλάσης Player) για το αντικείμενο URL και τοποθετείται σε αυτόν ένας ControllerListener (ακροατής) για να ελέγχεται. Αφού ο Player περάσει τις καταστάσεις realized και prefetched χωρίς προβλήματα είναι έτοιμος να ξεκινήσει. Ο Player ξεκινάει και έτσι το βίντεο αναπαράγαγεται. Στο παρακάτω σχήμα παρουσιάζεται η ροή της διαδικασίας που μόλις αναλύθηκε.




**Σχήμα 3**

### *2.4.3. Σχεδιασμός της διαδικασίας συρραφής εικόνων*

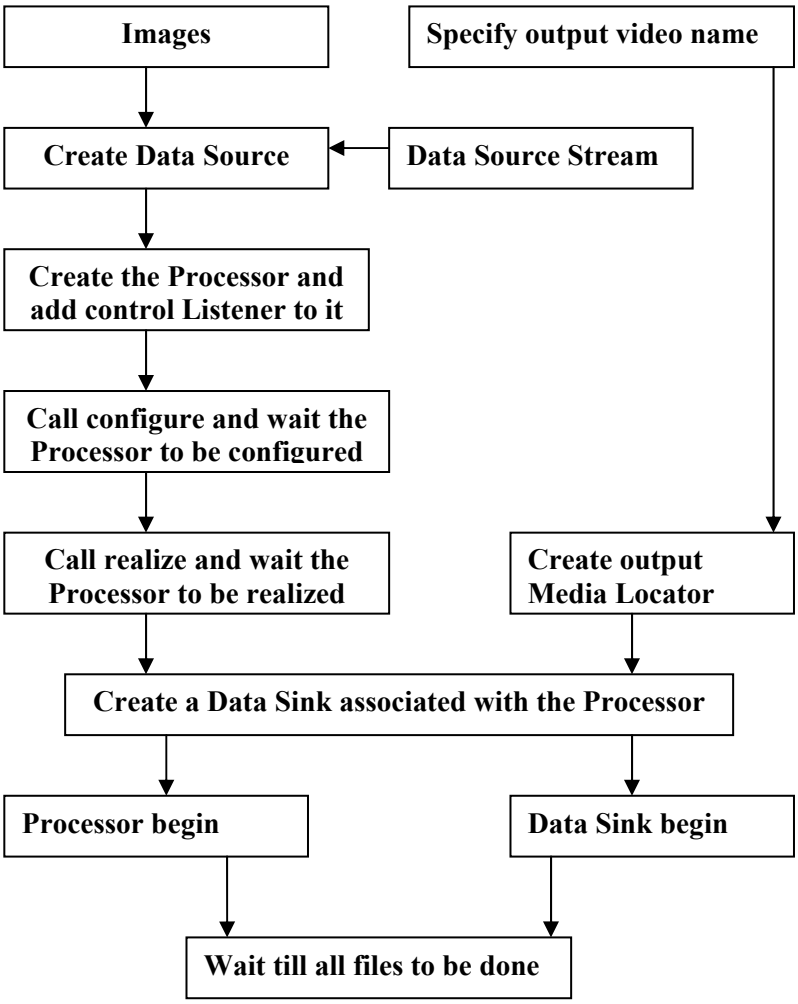
Με την χρήση της κλάσης Processor της JMF ένα αρχείο πολυμέσων σε μία εφαρμογή Java υπόκειται σε επεξεργασία. Ο Processor, εν αντιθέσει με τον Player, προσφέρει την δυνατότητα επεξεργασίας (π.χ.: αλλαγής του Format) του αρχείου εισόδου και επαναχρησιμοποίησης του αρχείου εξόδου μετά την επεξεργασία είτε με ένα νέο Processor είτε αποθηκεύοντας το στο δίσκο. Η λειτουργία που κάνει ο Processor είναι απλή όπως φαίνεται στο παρακάτω σχήμα.




**Σχήμα 4**

Όσον αφορά την συρραφή εικόνων αρχικά η εφαρμογή παίρνει το σύνολο των εικόνων που θα ενωθούν και το όνομα του αρχείου εξόδου. Στη συνέχεια δημιουργεί ένα DataSource (αντικείμενο της κλάσης DataSource) με παράμετρο τα ονόματα των αρχείων που θα ενωθούν και έναν Processor (αντικείμενο της κλάσης Processor). Αφού δημιουργηθεί ο Processor και προστεθεί σε αυτόν ένας ControllerListener, η εφαρμογή καλεί την λειτουργία Configure του Processor και περιμένει μέχρι ο

Processor να βρεθεί στην κατάσταση Configured. Ομοίως η εφαρμογή καλεί την λειτουργία Realize και περιμένει μέχρι ο Processor να βρεθεί στην κατάσταση Realized. Στην συνέχεια αφού ο Processor βρεθεί στην κατάσταση Realized η εφαρμογή δημιουργεί ένα DataSink το οποίο συσχετίζεται με τον Processor και παίρνει σαν παράμετρο τον MediaLocator (αντικείμενο της κλάσης MediaLocator) του αρχείου εξόδου. Η εφαρμογή στη συνέχεια ξεκινάει τον Processor και το DataSink. Έτσι τα δεδομένα από όλες τις εικόνες διαβάζονται και η εφαρμογή παράγει το αρχείο βίντεο στην προκαθορισμένη θέση στο δίσκο. Στο παρακάτω σχήμα παρουσιάζεται η ροή της διαδικασίας που μόλις αναλύθηκε.



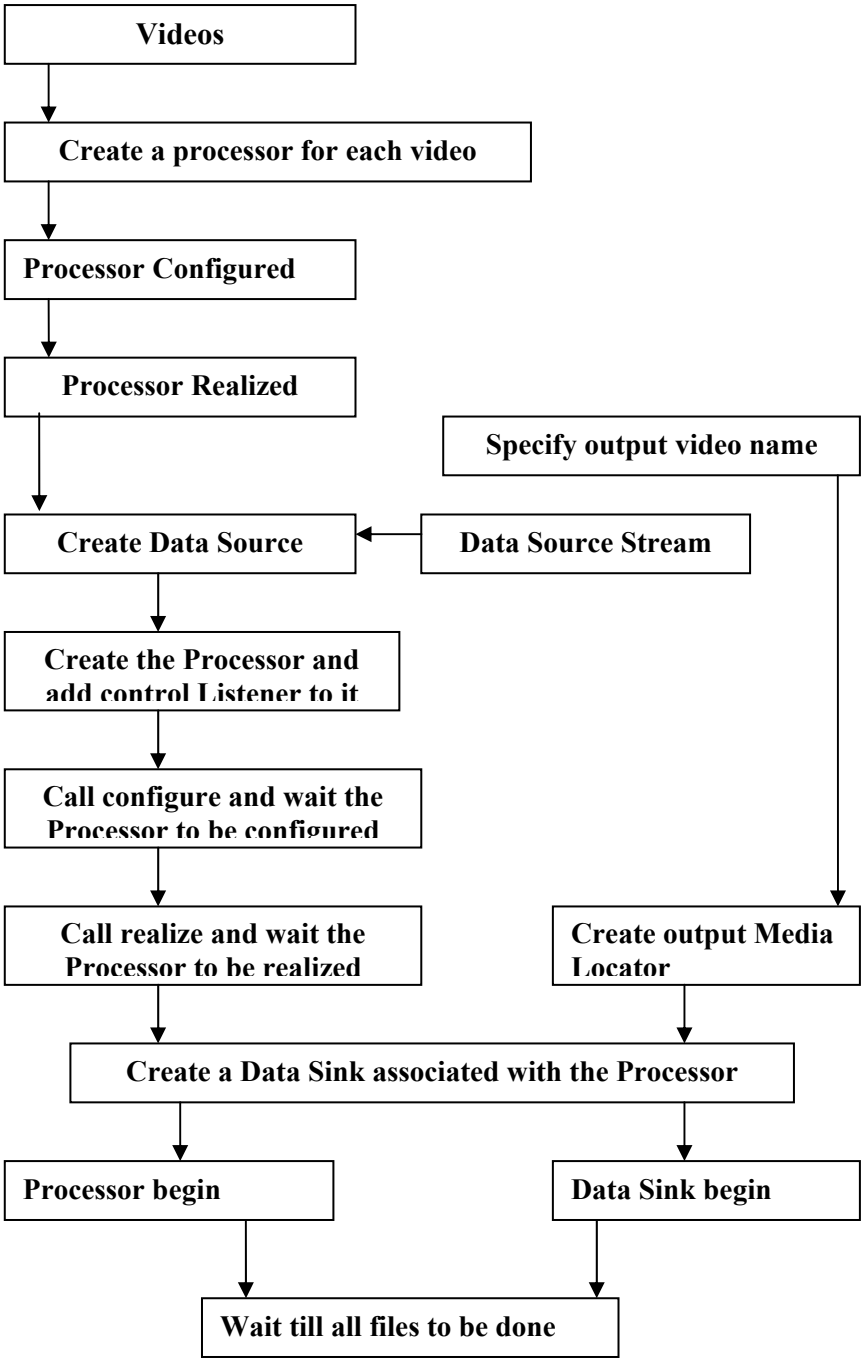

**Σχήμα 5**

### *2.4.4. Σχεδιασμός της διαδικασίας συρραφής βίντεο*

Όπως και στην διαδικασία συρραφής εικόνων, έτσι και στην διαδικασία συρραφής βίντεο θα χρησιμοποιηθεί η κλάση Processor της JMF. Με την κλάση

Processor είναι δυνατό να γίνουν οι κατάλληλοι χειρισμοί για την συρραφή των βίντεο.

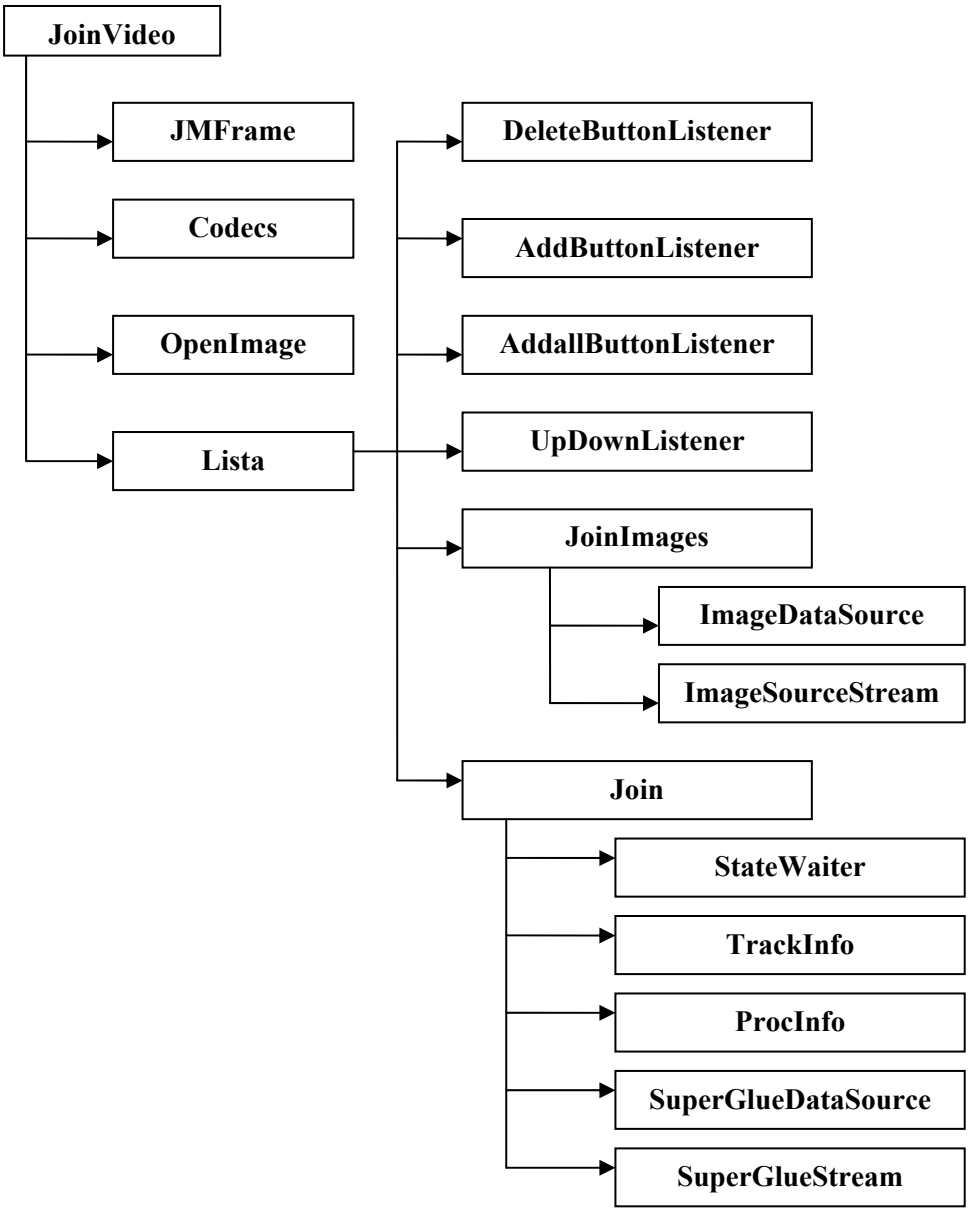
Στην περίπτωση της συρραφής βίντεο αρχικά δημιουργείται ένας Processor για κάθε βίντεο εισόδου. Αυτό γίνεται για ελεγχθεί αν είναι δυνατό να ενωθούν τα βίντεο. Στην συνέχεια η εφαρμογή βάζει τον Processor στην κατάσταση Configured για να κάνει τις αλλαγές στο Format του βίντεο, στην περίπτωση που αυτές χρειάζεται να γίνουν. Αφού γίνουν οι αλλαγές, η εφαρμογή τοποθετεί τον Processor στην κατάσταση Realized. Έτσι μπορεί πλέον να ξανακτίσει τα αρχεία με τις αλλαγές. Η διαδικασία αυτή γίνεται για όλα βίντεο. Στην συνέχεια η εφαρμογή δημιουργεί ένα DataSource με τα νέα αρχεία και μετά δημιουργεί ένα Processor. Αφού δημιουργηθεί ο Processor και προστεθεί σε αυτόν ένας ControllerListener, η εφαρμογή καλεί την λειτουργία Configure του Processor και περιμένει μέχρι ο Processor να βρεθεί στην κατάσταση Configured. Ομοίως η εφαρμογή καλεί την λειτουργία Realize και περιμένει μέχρι ο Processor να βρεθεί στην κατάσταση Realized. Στην συνέχεια αφού ο Processor βρεθεί στην κατάσταση Realized η εφαρμογή δημιουργεί ένα DataSink το οποίο συσχετίζεται με τον Processor και παίρνει σαν παράμετρο τον MediaLocator του αρχείου εξόδου. Η εφαρμογή στη συνέχεια ξεκινάει τον Processor και το DataSink. Έτσι τα δεδομένα από όλα τα βίντεο διαβάζονται και η εφαρμογή παράγει το αρχείο βίντεο στην προκαθορισμένη θέση στο δίσκο. Στο παρακάτω σχήμα παρουσιάζεται η ροή της διαδικασίας που μόλις αναλύθηκε.


Videos
Create a processor for each video
Processor Configured
Processor Realized
Specify output video name
Create Data Source
Data Source Stream
Create the Processor and add control Listener to it
Call configure and wait the Processor to be configured
Call realize and wait the Processor to be realized
Create output Media Locator
Create a Data Sink associated with the Processor
Processor begin
Data Sink begin
Wait till all files to be done

**Σχήμα 6**

# 3. Υλοποίηση

Για την υλοποίηση της εφαρμογής που παρουσιάζεται ήταν απαραίτητη η δημιουργία πολλών κλάσεων, έτσι θεωρήθηκε σωστό να γίνει παρουσίαση της δομής των κλάσεων με τέτοιο τρόπο ώστε να γίνει κατανοητή η μεταξύ τους επικοινωνία, σε μια προσπάθεια να αποδοθεί ο τρόπος κλήσης και η μεταξύ τους εξάρτηση. Στο γράφημα που ακολουθεί επιχειρείται η προσέγγιση της δομής του προγράμματος με δενδροειδή μορφή, ξεκινώντας από τη βασική κλάση της εφαρμογής την JoinVideo. Στην συνέχεια γίνεται αναλυτική αναφορά για το ρόλο των σημαντικότερων κλάσεων μέσα στην εφαρμογή. Αναλύοντας και παρουσιάζοντας ξεχωριστά κάθε λειτουργία της εφαρμογής, εξηγείται ο τρόπος αλληλεπίδρασης μεταξύ των κλάσεων και ο λόγος ύπαρξης τους.




**Σχήμα 7**

## *3.1. Εκτέλεση της εφαρμογής*

Στην ενότητα αυτή παρουσιάζεται συνοπτικά ο ρόλος των περισσοτέρων κλάσεων που αποτελούν την εφαρμογή κατά την εκτέλεση αυτής. Αναλυτικότερη περιγραφή του κώδικα των σημαντικότερων κλάσεων από τις οποίες αποτελείται η εφαρμογή θα παρουσιαστεί σε επόμενες ενότητες.

Για να ξεκινήσει η εφαρμογή γίνεται κλήση της κλάσης JoinVideo η οποία αποτελεί την κύρια κλάση της εφαρμογής. Η κλάση JoinVideo, με την βοήθεια και άλλων κλάσεων (κυρίως της κλάσης Lista), δημιουργεί το γραφικό περιβάλλον (GUI) της εφαρμογής και αποτελεί έτσι το βασικό δίαυλο επικοινωνίας μεταξύ του χρήστη

και της εφαρμογής. Επίσης μέσα από την κλάση JoinVideo γίνεται κλήση των υπολοίπων βασικών κλάσεων όποτε χρειάζεται.

Οι κλάσεις από τις οποίες αποτελείται η εφαρμογή μπορούν να διαχωριστούν σε δύο ομάδες. Στην πρώτη ομάδα ανήκουν οι κλάσεις που παράγουν το γραφικό περιβάλλον και διαχειρίζονται την επικοινωνία του χρήστη με την εφαρμογή. Στην δεύτερη ομάδα κλάσεων ανήκουν οι κλάσεις με τις οποίες η εφαρμογή προχωρεί στην συρραφή των αρχείων (βίντεο ή εικόνων).

Με το ξεκίνημα της εφαρμογής εμφανίζεται στον χρήστη η πλατφόρμα εργασίας. Στο στάδιο αυτό η εφαρμογή χρησιμοποιεί την πρώτη ομάδα κλάσεων (όπως αναφέρεται παραπάνω). Στην πλατφόρμα εργασίας της εφαρμογής ο χρήστης πραγματοποιεί όλες τις ενέργειες που επιθυμεί να κάνει και του προσφέρει η εφαρμογή πριν προχωρήσει στην συρραφή των αρχείων. Κυρίως οι κλάσεις JoinVideo και Lista δίνουν στην εφαρμογή το γραφικό περιβάλλον. Σε αυτό ο χρήστης μπορεί να επιλέξει τα αρχεία που επιθυμεί να συρράψει, να δώσει τις κατάλληλες πληροφορίες για αυτά, να τα βάλει στη σειρά που επιθυμεί και να προσθέσει ή να αφαιρέσει αρχεία. Η κλάση OpenImage παρέχει την δυνατότητα στον χρήστη να εμφανίσει μία εικόνα, ενώ η κλάση JMFrame δίνει στο χρήστη την δυνατότητα να αναπαράγει ένα βίντεο.

Αφού ο χρήστης τελειώσει με την επιλογή των αρχείων και είναι έτοιμος να προχωρήσει στην συρραφή των αρχείων, την σκυτάλη στην εφαρμογή αναλαμβάνει η δεύτερη ομάδα κλάσεων. Κατά την συρραφή των αρχείων έχουμε δύο περιπτώσεις ανάλογα με τον τύπο (βίντεο, εικόνες) των αρχείων που πρόκειται να ενωθούν. Στην περίπτωση που ο χρήστης επιλέξει να συρράψει εικόνες, η όλη διαδικασία πραγματοποιείται με την χρήση της κλάσης JoinImages και των υπολοίπων κλάσεων που αυτή χρησιμοποιεί, ενώ αν ο χρήστης επιλέξει να συρράψει βίντεο, η όλη διαδικασία πραγματοποιείται με την χρήση της κλάσης Join και των υπολοίπων κλάσεων που αυτή χρησιμοποιεί.

Από τα παραπάνω γίνεται αντιληπτό ότι η περισσότερη «δουλειά» στην εφαρμογή γίνεται από την δεύτερη ομάδα κλάσεων, αφού ουσιαστικά με την χρήση των κλάσεων της δεύτερης ομάδας γίνεται η συρραφή των αρχείων, ενώ με την χρήση των κλάσεων της πρώτης ομάδας απλά δημιουργείται το γραφικό περιβάλλον και υλοποιείται η επικοινωνία του χρήστη με την εφαρμογή.

Εφόσον η διαδικασία της συρραφής τελειώσει χωρίς προβλήματα ο χρήστης μπορεί, με την χρήση της κλάσης Codecs, να παρουσιάσει τα χαρακτηριστικά των

Format των βίντεο, ή με την χρήση της κλάσης JMFrame, να παρακολουθήσει το βίντεο το οποίο έδωσε η συρραφή των αρχείων.

Παρακάτω στις ενότητες που ακολουθούν περιγράφονται οι κλάσεις και οι μέθοδοι που χρησιμοποιήθηκαν στην εφαρμογή. Η ανάλυση χωρίζεται σε τρεις ενότητες. Στην πρώτη ενότητα περιγράφονται οι κλάσεις και οι μέθοδοι που δημιουργούν το γραφικό περιβάλλον (GUI) της εφαρμογής και γενικότερα είναι υπεύθυνες για την όλη λειτουργία της εφαρμογής αλλά και για την επικοινωνία αυτής με το χρήστη. Στη δεύτερη ενότητα περιγράφονται οι κλάσεις και οι μέθοδοι που υλοποιούν την διαδικασία της συρραφής εικόνων. Τέλος στην τρίτη ενότητα περιγράφονται οι κλάσεις και οι μέθοδοι που υλοποιούν την διαδικασία της συρραφής βίντεο.

## 3.2. Δημιουργία του GUI της εφαρμογής

### 3.2.1. Κλάση JoinVideo

Η εκκίνηση της εφαρμογής γίνεται με την κλήση από την Java της κλάσης JoinVideo. Η κλάση JoinVideo είναι υποκλάση της κλάσης JFrame (extends JFrame). Το συστατικό (κλάση) JFrame αποτελεί το αρχικό συστατικό – υποδοχέα σε μία παραθυρική εφαρμογή. Έτσι η JoinVideo κληρονομεί όλα τα χαρακτηριστικά της JFrame τα οποία και χρησιμοποιεί για να δημιουργήσει το κύριο παράθυρο της εφαρμογής. Στο παράθυρο αυτό εμφανίζεται το γραφικό περιβάλλον της εφαρμογής. Στον υποδοχέα JFrame λοιπόν, τοποθετείται με τον διαχειριστή διάταξης BorderLayout ένα πλαίσιο JPanel. Σε αυτό, με τον διαχειριστή διάταξης GridLayout, τοποθετούνται δύο JLayeredPane. Στο πρώτο JLayeredPane τοποθετούνται με τον διαχειριστή διάταξης GridLayout 7 JPanel τα οποία περιέχουν κουμπιά (JButton), ετικέτες (JLabel), και πεδία κειμένου (JTextField). Στο δεύτερο JLayeredPane τοποθετείται το JPanel που δίνει η κλάση Lista. Επίσης στον αρχικό υποδοχέα (JFrame) τοποθετείται η γραμμή μενού (MenuBar) η οποία και δημιουργείται από την μέθοδο createMenuBar().

Ιδιαίτερο ενδιαφέρον παρουσιάζουν τα δύο κουμπιά (Browse) τα οποία χρησιμοποιούνται για να επιλέξει ο χρήστης τα αρχεία που επιθυμεί να συρράψει. Το ένα κουμπί είναι για την επιλογή των εικόνων και το άλλο για την επιλογή των βίντεο. Στα κουμπιά αυτά έχουν προστεθεί ακροατές (actionListener) οι οποίοι

δημιουργούν γεγονότα (ActionEvent) με το πάτημα του κουμπιού. Στο ActionEvent λοιπόν που δημιουργείται με το πάτημα του κουμπιού, εμφανίζεται ένα παράθυρο διαλόγου (JFileChooser) από το οποίο γίνεται η επιλογή των αρχείων. Στο σημείο αυτό έχουν γίνει οι κατάλληλες ρυθμίσεις έτσι ώστε ο χρήστης να έχει την δυνατότητα να επιλέξει είτε ένα μοναδικό αρχείο κάθε φορά είτε ένα ολόκληρο φάκελο με αρχεία. Το όνομα και η διαδρομή του αρχείου ή των αρχείων, που θα επιλεγούν από το χρήστη, τοποθετούνται σε ένα πίνακα τύπου File με την χρήση των μεθόδων getSelectedFile() και getPath(). Στην περίπτωση που ο χρήστης επιλέξει ένα ολόκληρο φάκελο με αρχεία, η επιλογή των αρχείων από το φάκελο γίνεται με την χρήση της μεθόδου listFiles(). Τέλος και από τα δύο αυτά κουμπιά γίνεται κλήση της μεθόδου combos() στην περίπτωση συρραφής βίντεο ή της μεθόδου combos1() στην περίπτωση συρραφής εικόνων, με παράμετρο και στις δύο μεθόδους τον πίνακα με τα επιλεγμένα αρχεία.

```
browse.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
        if(currentDirectory==null){
                fc=new JFileChooser("c:/");
        }else {
                fc=new JFileChooser(currentDirectory);
        }
          fc.setFileHidingEnabled(true);
                fc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
      fc.setMultiSelectionEnabled(false);
      fc.setDragEnabled(true);

           int returnVal = fc.showOpenDialog(JoinVideo.this);
     if (returnVal == JFileChooser.APPROVE_OPTION) {
                direct=fc.getSelectedFile();
                path.setText(""+direct.getPath());
                currentDirectory=direct;

          if (direct.isDirectory()){
                arxeio=direct.listFiles();
                int m=arxeio.length;

                                for(int i=0; i<arxeio.length; i++){
                        if(arxeio[i].isDirectory()){
                                m--;
                        }
                }
                files=new File[m];
                int k=0;
                for(int i=0; i<arxeio.length; i++){

                        if(arxeio[i].isDirectory()){
                        }
                        else{files[k]=arxeio[i];
                        k++;
```

```
                        }
                }
        }else{
            files=new File[1];
                files[0]=direct;
                }
                combos(files);
        }

        }}
        );
```

**Πίνακας 2**

Οι μέθοδοι combos() και combos1() δίνουν στην κλάση Lista τα στοιχεία για την συρραφή των αρχείων. Επειδή δεν μπορεί να γίνει συρραφή βίντεο και εικόνων, παρά μόνο βίντεο με βίντεο και εικόνων με εικόνων, οι μέθοδοι προχωρούν στις παρακάτω ενέργειες. Αφού ο χρήστης επιλέξει πρώτη φορά αρχεία, οι μέθοδοι τα προσθέτουν στην κλάση Lista. Αν όμως στην συνέχεια ο χρήστης επιλέξει να προσθέσει και άλλα αρχεία, τότε αν αυτά είναι του ίδιου τύπου (βίντεο ή εικόνες), με αυτά που είχε επιλέξει την προηγούμενη φορά, προστίθενται στα προηγούμενα. Αν όμως τα αρχεία δεν είναι ίδιου τύπου με τα προηγούμενα τότε αφαιρούνται όλα τα προηγούμενα αρχεία και προστίθενται στην κλάση Lista μόνο τα νέα αρχεία.

```
 void combos(File []arxeio){
        if(type=="icon"){
                lista.comboMovies.removeAllItems();
                  path1.setText("Folder with the images      ");
                  lista.listModel.removeAllElements();
                  lista.go.setEnabled(false);
                  lista.addButton.setEnabled(false);
                  lista.addallButton.setEnabled(false);
  }
  type="video";
  lista.type="video";
  this.arxeio=arxeio;
  for(int i=0;i<arxeio.length;i++){
  lista.comboMovies.addItem(arxeio[i].getPath());
 }
}

   void combos1(File []arxeio1){
        if(type=="video"){
                lista.comboMovies.removeAllItems();
                  path.setText("Folder with the videos      ");
                  lista.listModel.removeAllElements();
                  lista.go.setEnabled(false);
                  lista.addButton.setEnabled(false);
                  lista.addallButton.setEnabled(false);
  }
  type="icon";
  lista.type="icon";
```

```
    this.arxeio1=arxeio1;
    for(int i=0;i<arxeio1.length;i++){
    lista.comboMovies.addItem(arxeio1[i].getPath());
   }
  }
```

**Πίνακας 3**

Το τρίτο κουμπί Browse έχει ένα ακροατή (actionListener) ο οποίος δημιουργεί ένα γεγονός (ActionEvent). Στο γεγονός που δημιουργείται με το πάτημα του κουμπιού εμφανίζεται ένα παράθυρο διαλόγου (JFileChooser) στο οποίο ο χρήστης επιλέγει το όνομα και την διαδρομή στον δίσκο του αρχείου βίντεο το οποίο θα δώσει η εφαρμογή με την συρραφή. Το όνομα και η διαδρομή στο δίσκο του αρχείο βίντεο αποθηκεύεται σε μία μεταβλητή τύπου String και προστίθενται στην κλάση Lista για να χρησιμοποιηθεί αργότερα από εκεί

Η επιλογή Open Video στο μενού File της εφαρμογής ανοίγει ένα παράθυρο διαλόγου (FileDialog) όπου ο χρήστης μπορεί να επιλέξει ένα αρχείο βίντεο που θέλει να εμφανίσει. Το όνομα και η διαδρομή του αρχείου αυτού στο δίσκο αποθηκεύονται σε μία μεταβλητή τύπου String. Στη συνέχεια καλείται η μέθοδος openFile() με παράμετρο τη μεταβλητή αυτή.

Στη μέθοδο openFile() δημιουργείται μία μεταβλητή τύπου URL για το βίντεο που επιθυμεί να εμφανίσει ο χρήστης, και ένας Player (αντικείμενο της κλάσης Player) για το URL αυτό. Τέλος αφού ο Player δημιουργηθεί χωρίς προβλήματα, καλείται η κλάση JMFrame με παραμέτρους τον Player και μία μεταβλητή τύπου String με το όνομα και την διαδρομή στο δίσκο του αρχείου βίντεο.

```
public void openFile(String filename) {
        String mediaFile = filename;
        Player player = null;

        URL url = null;
        try {

            if ((url = new URL(mediaFile)) == null) {
                Err("Can't build URL for " + mediaFile);
                return;
            }
            try {
                player = Manager.createPlayer(url);
            } catch (NoPlayerException e) {
                Err("Error: " + e);
            }
        } catch (MalformedURLException e) {
            Err("Error:" + e);
        } catch (IOException e) {
            Err("Error:" + e);
```

```
        }
        if (player != null) {
            this.filename = filename;
            JMFrame jmframe = new JMFrame(player, filename);
        }
    }
```

**Πίνακας 4**

Από τις επιλογές Open Image, Play outputfile και Codecs του μενού της εφαρμογής, γίνεται κλήση κλάσεων οι οποίες κάνουν διάφορες λειτουργίες. Οι κλάσεις αυτές, όπως και οι JMFrame και Lista, θα αναλυθούν στην συνέχεια.

### *3.2.2. Κλάση Lista*

Η κλάση Lista είναι υποκλάση της κλάσης JPanel (extends JPanel) και παράγει ένα πλαίσιο JPanel το οποίο και χρησιμοποιείται για την δημιουργία του γραφικού περιβάλλοντος της εφαρμογής. Ο αρχικός υποδοχέας λοιπόν στην συγκεκριμένη κλάση είναι το JPanel. Στο JPanel τοποθετούνται με την χρήση του διαχειριστή διάταξης BorderLayout τρία συστατικά. Το πρώτο συστατικό είναι ένα JPanel το οποίο περιέχει κουμπιά (JButton), τα οποία κάνουν διάφορες λειτουργίες. Το δεύτερο συστατικό είναι ένα πτυσσόμενο πλαίσιο (JComboBox) στο οποίο περιέχονται όλα τα αρχεία που έχει επιλέξει ο χρήστης για την συρραφή. Το τρίτο συστατικό είναι ένα JScrollPane το οποίο περιέχει μία λίστα (JList). Στη λίστα αυτή ο χρήστης μπορεί να προσθέσει αρχεία από το πτυσσόμενο πλαίσιο και στη συνέχεια να τους αλλάξει την σειρά ή να σβήσει μερικά από αυτά. Όλες αυτές οι λειτουργίες γίνονται με την χρήση των κουμπιών που βρίσκονται στο πρώτο συστατικό.

Στο κουμπί Go έχει τοποθετηθεί ένας ακροατής ο οποίος δημιουργεί ένα ActionEvent μόλις το κουμπί πατηθεί. Με το κουμπί Go αρχίζει η διαδικασία της συρραφής βίντεο. Πριν όμως αρχίσει η διαδικασία της συρραφής εξετάζεται αν πρόκειται για συρραφή βίντεο ή συρραφή εικόνων και αναλόγως γίνεται κλήση της κλάσης Join για βίντεο ή της κλάσης JoinImages για εικόνες. Επίσης εξετάζεται η ορθότητα των τιμών που έχει δώσει ο χρήστης στα πεδία κειμένου FrameRate, Width και Height.

Στα κουμπιά Add, Addall, Delete, MoveUp και MoveDown έχουν τοποθετηθεί ακροατές οι οποίοι καλούν τις κλάσεις AddButtonListener, AddallButtonListener, DeleteButtonListener και UpDownListener αντίστοιχα. Οι κλάσεις αυτές δημιουργούν ActionEvents τα οποία κάνουν τις ανάλογες ενέργειες.

Για παράδειγμα στην κλάση AddallButtonListener το ActionEvent τοποθετεί όλα τα αρχεία από το JComboBox στην JList και κάνει το κουμπί Go ενεργό στην περίπτωση όπου τα αρχεία είναι περισσότερα από ένα. Στο ActionEvent της κλάσης UpDownListener γίνεται χρήση της μεθόδου swap(). Η μέθοδος swap() αλλάζει την σειρά των αρχείων στην JList.

Τέλος στην JList έχει προστεθεί ένας ακροατής (ListSelectionListener) με τον οποίο και μέσω της μεθόδου valueChanged() ελέγχεται ο αριθμός των αρχείων που υπάρχουν στην JList και αναλόγως τα κουμπιά Go, Delete, MoveUp και MoveDown γίνονται ενεργά ή ανενεργά.

```
class Lista extends JPanel implements ListSelectionListener {

    public JList list;
    public DefaultListModel listModel;
    private static final String addString = "Add";
    private static final String addallString = "Add all";
    private static final String deleteString = "Delete";
    private static final String upString = "Move up";
    private static final String downString = "Move down";

    public JButton addButton;
    public JButton addallButton;
    private JButton deleteButton;
    private JButton upButton;
    private JButton downButton;
    public JComboBox comboMovies;
    String output="file:c:\\tmp\\outputfile.mov";
    public JButton go;
    Join join;
    JoinImages joinimages;
    public String type;
    float frameRate;
    int width,height;

    public Lista() {
        super(new BorderLayout());

        //Create and populate the list model.
        listModel = new DefaultListModel();

        //Create the list and put it in a scroll pane.
        list = new JList(listModel);
        list.setDragEnabled(true);
        list.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
        list.setSelectedIndex(0);
        list.addListSelectionListener(this);
        JScrollPane listScrollPane = new JScrollPane(list);

        //Create the list-modifying buttons.
        addButton = new JButton(addString);
        addButton.setActionCommand(addString);
        addButton.addActionListener(new AddButtonListener());
```

```
        addButton.setEnabled(false);

        addallButton = new JButton(addallString);
        addallButton.setActionCommand(addallString);
        addallButton.addActionListener(new AddallButtonListener());
        addallButton.setEnabled(false);

        deleteButton = new JButton(deleteString);
        deleteButton.setActionCommand(deleteString);
        deleteButton.addActionListener(new DeleteButtonListener());
                                    deleteButton.setEnabled(false);

        upButton = new JButton("Move up");
        upButton.setToolTipText("Move the currently selected list item higher.");
        upButton.setActionCommand(upString);
        upButton.addActionListener(new UpDownListener());
                                    upButton.setEnabled(false);

        downButton = new JButton("Move down");
        downButton.setToolTipText("Move the currently selected list item lower.");
        downButton.setActionCommand(downString);
        downButton.addActionListener(new UpDownListener());
                                    downButton.setEnabled(false);

        JPanel addPanel= new JPanel(new GridLayout(2,1));
        addPanel.add(addButton);
                                    addPanel.add(addallButton);

        JPanel upDownPanel = new JPanel(new GridLayout(2, 1));
        upDownPanel.add(upButton);
        upDownPanel.add(downButton);

        comboMovies=new JComboBox();
        comboMovies.setMaximumRowCount(6);

        go=new JButton("GO");
        go.setEnabled(false);
        go.setToolTipText("If you finished with the files click this button to join them.");

        JPanel buttonPane = new JPanel();
        buttonPane.add(go);
        buttonPane.add(addPanel);
        buttonPane.add(deleteButton);
        buttonPane.add(upDownPanel);
        comboMovies.setPreferredSize(new Dimension(350,24));
        listScrollPane.setPreferredSize(new Dimension(350,140));
        add(buttonPane, BorderLayout.PAGE_START);
        add(comboMovies, BorderLayout.CENTER);
        add(listScrollPane, BorderLayout.PAGE_END);

        comboMovies.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                    if(comboMovies.getSelectedItem()!=null){
             addButton.setEnabled(true);
             addallButton.setEnabled(true);
          }
        }});

        go.addActionListener(new ActionListener(){
```

```
        public void actionPerformed(ActionEvent e){

                JoinVideo.frr.setVisible(true);

                  int b=listModel.getSize();
                   String []listdata=new String[listModel.getSize()];
                   String []listdata1=new String[listModel.getSize()];
                   for(int i=0; i<b; i++){
                      list.setSelectedIndex(i);
                      listdata[i]="file:"+(String)list.getSelectedValue();
                      listdata1[i]=(String)list.getSelectedValue();
                     }
                if(type=="video"){

                        join=new Join(listdata,output);
      }
      else{
      if(JoinVideo.frameRate.getText().equals("")){
                frameRate=1;
      }else{
                try{
    frameRate=Float.parseFloat(JoinVideo.frameRate.getText());
     }catch (Exception ev){
      JoinVideo.frr.dispose();
      JoinVideo.Err("Error: "+ev);
      return;
      }
      if(frameRate>10 || frameRate<0.1){
        JoinVideo.frr.dispose();
       JoinVideo.Err("Error: The FrameRate is out of range!");
       return;
      }
                }
      if(JoinVideo.width.getText().equals("")){
                width=0;
      }else{
                try{
       width=Integer.parseInt(JoinVideo.width.getText());
     }catch (Exception ev){
      JoinVideo.frr.dispose();
      JoinVideo.Err("Error: "+ev);
      return;
      }
       if(width>1000 || width<10){
        JoinVideo.frr.dispose();
       JoinVideo.Err("Error: The width is out of range!");
       return;
      }
                }
      if(JoinVideo.height.getText().equals("")){
                height=0;
      }else{
                try{
       height=Integer.parseInt(JoinVideo.height.getText());
     }catch (Exception ev){
      JoinVideo.frr.dispose();
      JoinVideo.Err("Error: "+ev);
      return;
      }
```

```
        if(height>1000 || height<10){
         JoinVideo.frr.dispose();
        JoinVideo.Err("Error: The height is out of range!");
        return;
       }
                }
       joinimages=new JoinImages(width,height,frameRate,listdata1,output);
       }
       }});
}

class DeleteButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {


        ListSelectionModel lsm = list.getSelectionModel();
        int firstSelected = lsm.getMinSelectionIndex();
        int lastSelected = lsm.getMaxSelectionIndex();
        listModel.removeRange(firstSelected, lastSelected);


            if (firstSelected == listModel.getSize()) {
            //Removed item in last position.
                firstSelected--;
            }
            list.setSelectedIndex(firstSelected);

    }
}


class AddButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {


        int index = list.getSelectedIndex();
        int size = listModel.getSize();


        if (index == -1 || (index+1 == size)) {
            listModel.addElement(comboMovies.getSelectedItem());
            list.setSelectedIndex(size);
        } else {
            listModel.insertElementAt(comboMovies.getSelectedItem(), index+1);
            list.setSelectedIndex(index+1);
        }
    }
}


class AddallButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
 int n=0;
                                for(int i=0; i<comboMovies.getItemCount(); i++){
            listModel.addElement(comboMovies.getItemAt(i));
         n++;
         }
                                        if (n>1){
                                                go.setEnabled(true);
```

```
                                                }

    }
}

class UpDownListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {

        int moveMe = list.getSelectedIndex();

        if (e.getActionCommand().equals(upString)) {
        //UP BUTTON
            if (moveMe != 0) {
                swap(moveMe, moveMe-1);
                list.setSelectedIndex(moveMe-1);
                list.ensureIndexIsVisible(moveMe-1);
            }
        } else {
        //DOWN BUTTON
            if (moveMe != listModel.getSize()-1) {
                swap(moveMe, moveMe+1);
                list.setSelectedIndex(moveMe+1);
                list.ensureIndexIsVisible(moveMe+1);
            }
        }
    }
}

//Swap two elements in the list.
private void swap(int a, int b) {
    Object aObject = listModel.getElementAt(a);
    Object bObject = listModel.getElementAt(b);
    listModel.set(a, bObject);
    listModel.set(b, aObject);
}
//Listener method for list selection changes.
public void valueChanged(ListSelectionEvent e) {
        if (list.getSelectedIndex() == -1) {
            deleteButton.setEnabled(false);
            upButton.setEnabled(false);
            downButton.setEnabled(false);

        } else if (list.getSelectedIndices().length > 1) {
            deleteButton.setEnabled(true);
            upButton.setEnabled(false);
            downButton.setEnabled(false);

        } else {
            deleteButton.setEnabled(true);
            upButton.setEnabled(true);
            downButton.setEnabled(true);
        }
        if (listModel.getSize()>=2){
                go.setEnabled(true);
                        }
                else{
                go.setEnabled(false);
                upButton.setEnabled(false);
        downButton.setEnabled(false);
```

```
                }
    }
}
```

**Πίνακας 5**

### *3.2.3. Κλάση JMFrame*

Στην μέθοδο openFile() της κλάσης JoinVideo γίνεται κλήση του κατασκευαστή της κλάσης JMFrame με ορίσματα έναν Player (αντικείμενο της κλάσης Player) και μία μεταβλητή τύπου String με περιεχόμενο το όνομα και την διαδρομή του αρχείου βίντεο στο δίσκο. Η κλάση JMFrame είναι υποκλάση της JFrame. Στον υποδοχέα JFrame λοιπόν της κλάσης ορίζεται διαχειριστής διάταξης ο BorderLayout. Στη συνέχεια γίνεται καταχώριση ενός ακροατή (ControllerListener) για τα γεγονότα του Player και καλείται η μέθοδος realize() της κλάσης Player για να περάσει ο Player στην κατάσταση Realized. Αφού ο Player βρεθεί στην κατάσταση realized, καλείται η μέθοδος prefetch() της κλάσης Player. Όταν ο Player βρεθεί στην κατάσταση prefetched, τότε με την μέθοδο getVisualComponent() αντλείται από τον Player το συστατικό όπου «παίζει» το βίντεο και τοποθετείται στο JFrame. Επίσης με τη μέθοδο getControlPanelComponent αντλείται από τον Player το συστατικό το οποίο περιέχει τα κουμπιά ελέγχου και προστίθεται και αυτό με τη σειρά του στο JFrame. Τέλος με την κλήση της μεθόδου start() της κλάσης Player ο Player ξεκινάει και το βίντεο παρουσιάζεται στον χρήστη. Αφού η αναπαραγωγή του βίντεο τελειώσει, με τη μέθοδο setMediaTime() της κλάσης Player τοποθετείται ο δείκτης της ροής του βίντεο στην αρχή.

Όλα τα παραπάνω δημιουργούνται και λειτουργούν σωστά αφού ο Player καταφέρει να αποκωδικοποιήσει το βίντεο και εμφανίζονται σε ένα νέο παράθυρο με τίτλο το όνομα και τη διαδρομή του αρχείου βίντεο στο δίσκο. Στο παράθυρο αυτό γίνεται η αναπαραγωγή του βίντεο και περιέχονται τα βασικά κουμπιά ελέγχου με τα οποία ο χρήστης μπορεί να ελέγξει τη ροή αναπαραγωγής του βίντεο. Αν δημιουργηθεί κάποιο πρόβλημα κατά το άνοιγμα κάποιου βίντεο, τότε εμφανίζεται ένα μήνυμα στο χρήστη με περιεχόμενο την αιτία του προβλήματος.

```
class JMFrame extends JFrame implements ControllerListener {
    Player mplayer;
    Component visual = null;
    Component control = null;
    int videoWidth = 0;
    int videoHeight = 0;
    int controlHeight = 30;
```

```
    int insetWidth = 10;
    int insetHeight = 30;
    boolean firstTime = true;

    public JMFrame(Player player, String title) {
          super(title);
          getContentPane().setLayout( new BorderLayout() );
          setSize(320, 10);
          setLocation(50, 50);
          setVisible(true);
          mplayer = player;
          mplayer.addControllerListener((ControllerListener) this);
          mplayer.realize();

                addWindowListener( new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                mplayer.close();
              }
          } );
    }
    public void controllerUpdate(ControllerEvent ce) {
          if (ce instanceof RealizeCompleteEvent) {
              mplayer.prefetch();
          } else if (ce instanceof PrefetchCompleteEvent) {
              if (visual != null)
                    return;
              if ((visual = mplayer.getVisualComponent()) != null) {
                    Dimension size = visual.getPreferredSize();
                    videoWidth = size.width;
                    videoHeight = size.height;
                    getContentPane().add("Center", visual);
              } else
                    videoWidth = 320;
              if ((control = mplayer.getControlPanelComponent()) != null) {
                    controlHeight = control.getPreferredSize().height;
                    getContentPane().add("South", control);
              }
              setSize(videoWidth + insetWidth,
                      videoHeight + controlHeight + insetHeight);
              validate();
              mplayer.start();
          } else if (ce instanceof EndOfMediaEvent) {
              mplayer.setMediaTime(new Time(0));
          }
  }
}
```

**Πίνακας 6**

### *3.2.4. Κλάση OpenImage*

Η επιλογή Open Image στο μενού File της εφαρμογής ανοίγει ένα παράθυρο διαλόγου, όπου και επιλέγεται η εικόνα την οποία ο χρήστης θέλει να εμφανίσει και στη συνέχεια καλεί τον κατασκευαστή της κλάσης OpenImage με παράμετρο το όνομα και την διαδρομή της εικόνας στο δίσκο. Η κλάση OpenImage είναι υποκλάση

της κλάσης JFrame. Στον υποδοχέα JFrame της κλάσης OpenImage λοιπόν, μέσα σε ένα JScrollPane τοποθετείται το JPanel το οποίο περιέχει την εικόνα. Με την χρήση της κλάσης ImageIcon τοποθετείται η εικόνα στο JPanel. Η ImageIcon θα μπορούσε να πάρει παράμετρο το όνομα και τη διαδρομή της εικόνας στο δίσκο, αλλά προτιμήθηκε, επειδή έτσι υποστηρίζονται εικόνες περισσότερων τύπων, να δοθεί παράμετρος στην ImageIcon το αντικείμενο BufferedImage της εικόνας. Η εικόνα μετατρέπεται σε BufferedImage μέσω της μεθόδου read() της κλάσης ImageIO, με παράμετρο την μεταβλητή τύπου File του ονόματος και της διαδρομής της εικόνας στο δίσκο. Αφού η εικόνα διαβαστεί χωρίς προβλήματα, τότε δημιουργείται ένα νέο παράθυρο με περιεχόμενο την εικόνα και τίτλο το όνομα και την διαδρομή της εικόνας στο δίσκο, αλλιώς εμφανίζεται στο χρήστη μήνυμα με περιεχόμενο την αιτία του προβλήματος.

```
class OpenImage extends JFrame {
	BufferedImage img;

	public OpenImage(String filename) {
		super("Image: "+filename);

		try {
	  img = ImageIO.read(new File(filename));
	} catch (Exception e) {
	  JoinVideo.Err("Error: "+e);
	 return;
		}
try{
	 getContentPane().add(new JScrollPane(new JLabel(new ImageIcon(img))));
	 }catch (Exception e){
		JoinVideo.Err("Error: "+e);
		return;
	 }
		setSize(500,400 );
		setLocationRelativeTo(null);
	   setLocation(50, 50);
		setVisible(true);
 }
}
```

**Πίνακας 7**

### *3.2.5. Κλάση Codecs*

Η επιλογή Codecs στο μενού Options της εφαρμογής καλεί τον κατασκευαστή της κλάσης Codecs. Η κλάση Codecs είναι υποκλάση της JFrame. Στον υποδοχέα JFrame λοιπόν της κλάσης, τοποθετείται ένα JScrollPane μέσα στο οποίο τοποθετείται ένα JPanel. Στο τελευταίο, με διαχειριστή διάταξης GridLayout

τοποθετούνται JPanel κάθε ένα από τα οποία περιέχει JLabel με περιεχόμενο τα χαρακτηριστικά των Formats των βίντεο που χρησιμοποιήθηκαν στην συρραφή αλλά και του βίντεο εξόδου της εφαρμογής. Τα χαρακτηριστικά αυτά η κλάση τα αντλεί από την κλάση Join όπου και έχουν αποθηκευτεί σε μεταβλητές τύπου String. Στην περίπτωση που ο χρήστης δεν έχει προχωρήσει στην συρραφή των βίντεο, δεν υπάρχουν δηλαδή στοιχεία για εμφάνιση, τότε στο παράθυρο που δημιουργείται αντί για τα χαρακτηριστικά εμφανίζεται ένα μήνυμα.

```
class Codecs extends JFrame {
  public Codecs() {
          super("Video Codecs");
          getContentPane().setLayout( new BorderLayout() );
          setSize(400,300 );
          setLocation(50, 50);
          setVisible(true);
          JPanel pp=new JPanel();
          JScrollPane scroll = new JScrollPane(pp);
          getContentPane().add("Center", scroll);
          if(Join.aud==null){
          JLabel no=new JLabel("You must first join the videos and then you will see
the codecs!");
          pp.add(no);
          } else {
                    int k=(Join.aud.length+1)*3;
                    JPanel mini=new JPanel(new GridLayout(k,0));
          for(int i=0;i<Join.aud.length; i++){
          JLabel  ccc=new JLabel("File:"+(i+1));
          JLabel  aaa=new JLabel("Audio codec: "+Join.aud[i]);
          JLabel  bbb=new JLabel("Video codec: "+Join.vid[i]);
          mini.add(ccc);
          mini.add(aaa);
          mini.add(bbb);
                                        }
          mini.add(new JLabel("Output File"));
          mini.add(new JLabel("Audio codec: "+Join.outAud));
          mini.add(new JLabel("Video codec: "+Join.outVid));
                                                  pp.add(mini);
                    }
 }
}
```

**Πίνακας 8**

## *3.3. Συρραφή εικόνων*

### *3.3.1. Κλάση JoinImages*

Με την κλάση JoinImages υλοποιείται η διαδικασία της συρραφής εικόνων. Ο κατασκευαστής (Constructor) της JoinImages καλείται από την κλάση Lista με τα παρακάτω ορίσματα: ένα πίνακα τύπου String με τα ονόματα και την διαδρομή στο

δίσκο των εικόνων που πρόκειται να συρραφούν, μία μεταβλητή τύπου String με το όνομα και την διαδρομή στο δίσκο του αρχείου (βίντεο) εξόδου, μία μεταβλητή τύπου Float για το FrameRate του βίντεο και δύο μεταβλητές τύπου int για τις διαστάσεις του βίντεο. Αρχικά δημιουργείται ένας MediaLocator (αντικείμενο της κλάσης MediaLocator) για κάθε αρχείο (εικόνα) αλλά και για το αρχείο (βίντεο) εξόδου με την χρήση της μεθόδου createMediaLocator(). Τέλος στον κατασκευαστή της κλάσης καλείται η μέθοδος doIt() η οποία και ελέγχει την ροή της διαδικασίας συρραφής.

Η μέθοδος doIt() ξεκινάει με ένα απλό έλεγχο για τις εικόνες εισόδου, ελέγχει δηλαδή αν οι εικόνες μπορούν να διαβαστούν και να μετατραπούν σε BufferedImage. Στη συνέχεια δημιουργείται ένα αντικείμενο της κλάσης ImageDataSource και ένας Processor (αντικείμενο της κλάσης Processor) ο οποίος παίρνει παράμετρο το αντικείμενο της κλάσης ImageDataSource. Με την χρήση των μεθόδων configure() της κλάσης Processor και waitForState() ο Processor μεταβαίνει στην κατάσταση configured. Στον Processor τίθεται ContentDescriptor (setContentDescriptor()) η έξοδος της μεθόδου fileExtToCD() η οποία παίρνει παράμετρο τον MediaLocator του αρχείου εξόδου και επιστρέφει τον ContentDescriptor για το αρχείο αυτό. Δηλαδή αν το αρχείο εξόδου έχει κατάληξη ".mov" τότε ο ContentDescriptor στον Processor τίθεται ίσος με QUICKTIME. Μετά γίνεται έλεγχος για τα Format που υποστηρίζει ο Processor (getSupportedFormats()) σύμφωνα με τον ContentDescriptor και τίθεται (setFormat()) στον Processor ένα από τα υποστηριζόμενα Format. Στην συνέχεια με τις μεθόδους realize() της κλάσης Processor και waitForState() ο Processor μεταβαίνει στην κατάσταση realized. Μετά με την χρήση της μεθόδου createDataSink() με παραμέτρους τον Processor και το MediaLocator του αρχείου εξόδου δημιουργείται ένα DataSink. Τέλος ο Processor και το DataSink ξεκινάνε με την χρήση της μεθόδου start(). Έτσι ουσιαστικά ξεκινάει η διαδικασία της συρραφής και η εγγραφή των δεδομένων στο αρχείο εξόδου. Αφού με την χρήση της μεθόδου waitForFileDone() διαγνωθεί ότι η διαδικασία συρραφής τελείωσε, το DataSink τερματίζεται με την μέθοδο close() και καλείται η μέθοδος ok() της κλάσης JoinVideo η οποία με ένα μήνυμα ειδοποιεί το χρήστη ότι η συρραφή τελείωσε επιτυχώς.

```
public boolean doIt(int width, int height, float frameRate, Vector inFiles,
MediaLocator outML) {

                    for(int i=0;i<inFiles.size();i++){
```

```
            String imageFile = (String)inFiles.elementAt(i);
                                        BufferedImage img = null;
try {
    img = ImageIO.read(new File(imageFile));
} catch (IOException e) {
}
                        try{
       if(img.getWidth()==0 || img.getHeight()==0){
                         er="Error: Wrong input image.";
                         return false;
                                }
                        } catch (Exception e){
                                er="Error: "+e+" \nWrong input image";
                                return false;

                                }
}

          ImageDataSource ids = new ImageDataSource(width, height, frameRate,
inFiles);

          Processor p;

          try {
              p = Manager.createProcessor(ids);
          } catch (Exception e) {
              er="Cannot create a processor from the data source.";
              return false;
          }

          p.addControllerListener(this);

          p.configure();
          if (!waitForState(p, p.Configured)) {
             er="Failed to configure the processor.";
              return false;
          }

          ContentDescriptor cd;

          if ((cd = fileExtToCD(outML.getRemainder())) == null) {

                                er="Couldn't figure out from the file extension the type of
output needed!";
                                return false;
          }

          p.setContentDescriptor(cd);

          TrackControl tcs[] = p.getTrackControls();
          Format f[] = tcs[0].getSupportedFormats();
          if (f == null || f.length <= 0) {
              er="The mux does not support the input format: " + tcs[0].getFormat();
              return false;
}

          tcs[0].setFormat(f[0]);

          p.realize();
```

```
        if (!waitForState(p, p.Realized)) {
            er="Failed to realize the processor.";
            return false;
        }

        //create a DataSink.
        DataSink dsink;
        if ((dsink = createDataSink(p, outML)) == null) {
            er="Failed to create a DataSink for the given output MediaLocator.";
            return false;
        }

        dsink.addDataSinkListener(this);
        fileDone = false;

        try {
            p.start();
            dsink.start();
        } catch (IOException e) {
            er="IO error during processing";
            return false;
        }

        // Wait for EndOfStream event.

        waitForFileDone();

        // Cleanup.
        try {
           dsink.close();
        } catch (Exception e) {}
        p.removeControllerListener(this);

JoinVideo.frr.setVisible(false);
JoinVideo.ok("Join has be done successfully!");

        return true;
    }
```

**Πίνακας 9**

Σε κάθε βήμα (όπως παρουσιάστηκε παραπάνω) της διαδικασίας συρραφής εικόνων, η οποία επιτελείται από την μέθοδο doIt(), γίνεται έλεγχος για τυχόν προβλήματα που μπορεί να παρουσιαστούν. Αν προκύψει κάποιο πρόβλημα η διαδικασία συρραφής σταματάει και εμφανίζεται μήνυμα στο χρήστη με την κλήση της μεθόδου Err() της κλάσης JoinVideo με περιεχόμενο το πρόβλημα που προέκυψε. Παρακάτω θα περιγραφούν οι περισσότερες μέθοδοι και κλάσεις οι οποίες χρησιμοποιήθηκαν στη μέθοδο doIt() για την διαδικασία συρραφής εικόνων.

Η μέθοδος createDataSink() παίρνει το DataSource από τον Processor με την χρήση της μεθόδου getDataOutput(). Στη συνέχεια δημιουργεί ένα DataSink (αντικείμενο της κλάσης DataSink) με το DataSource που πήρε από τον Processor και

τον MediaLocator του αρχείου εξόδου. Τέλος η μέθοδος επιστρέφει το DataSink που δημιούργησε.

Η λειτουργία που επιτελεί η μέθοδος waitForState() είναι να διακόψει προσωρινά τη ροή της διαδικασίας της συρραφής, έως ότου ο Processor περάσει στην κατάσταση που του ζητήθηκε. Παρόμοια και η μέθοδος waitForFileDone() σταματάει προσωρινά την ροή του προγράμματος (δηλαδή το πρόγραμμα δεν θα μπορεί να χρησιμοποιηθεί) μέχρι να ολοκληρωθεί η διαδικασία της συρραφής, δηλαδή μέχρι να ολοκληρωθεί η εγγραφή του αρχείου εξόδου στο δίσκο.

### *3.3.2. Κλάση ImageDataSource*

Η κλάση ImageDataSource αποτελεί το DataSource το οποίο χρησιμοποιεί ο Processor για να προχωρήσει στην συρραφή των εικόνων. Η κλάση ImageDataSource είναι υποκλάση της κλάσης PullBufferDataSource, οπότε και κληρονομεί όλα τα χαρακτηριστικά της PullBufferDataSource. Αρχικά στην κλάση δημιουργείται ένα αντικείμενο της κλάσης ImageSourceStream. Στη συνέχεια εφαρμόζονται οι μέθοδοι τις οποίες η κλάση κληρονομεί από την υπερκλάση της. Για παράδειγμα η μέθοδος connect() δημιουργεί μία σύνδεση με την πηγή (αρχείο εισόδου) ενώ η μέθοδος start() ξεκινάει την μεταφορά των δεδομένων από τα αρχεία εισόδου προς το DataSink (δηλαδή προς το αρχείο εξόδου). Η μέθοδος getStreams() παίρνει το σύνολο των Streams (ροή δεδομένων) τα οποία το DataSource διαχειρίζεται, και τα επιστρέφει. Στην προκειμένη περίπτωση το σύνολο των Streams είναι το αντικείμενο της κλάσης ImageSourceStream, έτσι η getStreams() επιστρέφει κάθε Stream της ImageSourceStream, το οποίο είναι τύπου PullBufferStream αφού η κλάση ImageSourceStream χρησιμοποιεί τη διασύνδεση (implements) PullBufferStream.

```
class ImageDataSource extends PullBufferDataSource {

        ImageSourceStream streams[];

        ImageDataSource(int width, int height, float frameRate, Vector images) {
            streams = new ImageSourceStream[1];
            streams[0] = new ImageSourceStream(width, height, frameRate, images);
  }

        public void setLocator(MediaLocator source) {
        }

        public MediaLocator getLocator() {
            return null;
```

```
        }

        public String getContentType() {
            return ContentDescriptor.RAW;
        }

        public void connect() {
        }

        public void disconnect() {
        }

        public void start() {
        }

        public void stop() {
        }

        /**
         * Return the ImageSourceStreams.
         */
        public PullBufferStream[] getStreams() {
            return streams;
        }

        public Time getDuration() {
            return DURATION_UNKNOWN;
        }

        public Object[] getControls() {
            return new Object[0];
        }

        public Object getControl(String type) {
            return null;
          }
    }
```

**Πίνακας 10**

### *3.3.3. Κλάση ImageSourceStream*

Η κλάση ImageSourceStream, όπως αναφέρθηκε προηγουμένως, αντιπροσωπεύει το σύνολο των Streams τα οποία το DataSource διαχειρίζεται. Πιο συγκεκριμένα η ImageSourceStream διαβάζει τα δεδομένα από κάθε εικόνα και τα μετατρέπει σε Buffer. Αυτά τα Buffer αποτελούν το Stream του DataSource. Ο κατασκευαστής της κλάσης παίρνει ορίσματα τις διαστάσεις που έδωσε ο χρήστης για το βίντεο, το FramRate και τις εικόνες που θα ενωθούν. Στον κατασκευαστή έχουμε την δημιουργία ενός RGB format το οποίο και θα χρησιμοποιηθεί για την δημιουργία του βίντεο. Η επιλογή του RGB format είναι υποχρεωτική επειδή κατά την μετατροπή των εικόνων σε Buffer, με την χρήση της κλάσης ImageToBuffer, το Buffer που δημιουργείται πρέπει να έχει RGB format. Στην συνέχεια αναλαμβάνει η μέθοδος

read(). Στη μέθοδο read() λοιπόν κάθε εικόνα μετατρέπεται σε Buffer με την χρήση της κλάσης ImageToBuffer. Επίσης γίνονται οι απαραίτητες ρυθμίσεις στο Buffer κάθε φορά που μία εικόνα μετατρέπεται σε Buffer. Για να μπορέσει να αποθηκευτεί το Buffer όλων των εικόνων μαζί, να δημιουργηθεί δηλαδή το βίντεο που θα περιέχει τις εικόνες, πρέπει όλες οι εικόνες να έχουν τις ίδιες διαστάσεις. Η αλλαγή των διαστάσεων των εικόνων, όταν αυτή χρειάζεται, γίνεται με την χρήση των μεθόδων getScaledInstance(), copy() και getDefaultConfiguration(). Τέλος στην κλάση εφαρμόζονται οι μέθοδοι τις οποίες κληρονομεί (implements) από την διασύνδεση (interface) PullBufferStream.

```
class ImageSourceStream implements PullBufferStream {
  ImageToBuffer bu;
  Buffer buff;
  long sequenceNumber = 0;
        Vector images;
        int width, height;
        float frameRate;
        RGBFormat rgb,r;
boolean rescale=true;

Dimension diastaseis;

        int nextImage = 0;      // index of the next image to be read.
        boolean ended = false;

        public ImageSourceStream(int width, int height, float frameRate, Vector
images) {
            this.width = width;
            this.height = height;
            this.images = images;
                        this.frameRate=frameRate;

                                    String imageFile = (String)images.elementAt(0);
                                   BufferedImage img = null;
try {
   img = ImageIO.read(new File(imageFile));
} catch (IOException e) {
}

if(height==0 || width==0){
        this.width=img.getWidth();
        this.height=img.getHeight();
        }

        diastaseis=new Dimension(this.width,this.height);
  bu=new ImageToBuffer();
  buff=bu.createBuffer(img,frameRate);
                                   r=(RGBFormat)buff.getFormat();
                rgb=new         RGBFormat(diastaseis,
              r.getMaxDataLength(),
              Format.byteArray,
              frameRate,
```

```
                r.getBitsPerPixel(),
                r.getRedMask(),
                r.getGreenMask(),
                r.getBlueMask(),
                r.getPixelStride(),
                r.getLineStride(),
                r.getFlipped(),
                r.getEndian());
        }

        public boolean willReadBlock() {
            return false;
        }

        public void read(Buffer buf) throws IOException {

            // Check if we've finished all the frames.
            if (nextImage >= images.size()) {
                buf.setEOM(true);
                buf.setOffset(0);
                buf.setLength(0);
                ended = true;
                return;
            }

            String imageFile = (String)images.elementAt(nextImage);
            nextImage++;

BufferedImage icon=null;
BufferedImage img = null;
try {
    img = ImageIO.read(new File(imageFile));
    } catch (IOException e) {
    }

bu=new ImageToBuffer();
GraphicsConfiguration gc=getDefaultConfiguration();
icon=getScaledInstance(img,this.width,this.height,gc);

buff=bu.createBuffer(icon,frameRate);
        buf.setData(buff.getData());
        buf.setOffset(buff.getOffset());
        buf.setLength(buff.getLength());
        buf.setFormat(buff.getFormat());
        buf.setFlags(buf.getFlags() | Buffer.FLAG_KEY_FRAME);
        long time = (long)(sequenceNumber * 1.0e9 / frameRate);
        buf.setTimeStamp(time);
        buf.setSequenceNumber(sequenceNumber++);
         }


private GraphicsConfiguration getDefaultConfiguration() {
    GraphicsEnvironment                                    ge                                    =
GraphicsEnvironment.getLocalGraphicsEnvironment();
    GraphicsDevice gd = ge.getDefaultScreenDevice();
    return gd.getDefaultConfiguration();
}

private BufferedImage copy(BufferedImage source, BufferedImage target) {
```

```
    Graphics2D g2 = target.createGraphics();
    g2.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
RenderingHints.VALUE_INTERPOLATION_BICUBIC);
    double scalex = (double) target.getWidth()/ source.getWidth();
    double scaley = (double) target.getHeight()/ source.getHeight();
    AffineTransform xform = AffineTransform.getScaleInstance(scalex, scaley);
    g2.drawRenderedImage(source, xform);
    g2.dispose();
    return target;
}

private BufferedImage getScaledInstance(BufferedImage image, int width, int height,
GraphicsConfiguration gc) {
    if(image.getWidth() != width || image.getHeight() != height) {
    if (gc == null)
        gc = getDefaultConfiguration();
    int transparency = image.getColorModel().getTransparency();
    return copy(image, gc.createCompatibleImage(width, height, transparency));
}
else {
          return image;
          }
}

          public Format getFormat() {
    return rgb;
          }

          public ContentDescriptor getContentDescriptor() {
              return new ContentDescriptor(ContentDescriptor.RAW);
          }

          public long getContentLength() {
              return 0;
          }

          public boolean endOfStream() {
              return ended;
          }

          public Object[] getControls() {
              return new Object[0];
          }

          public Object getControl(String type) {
              return null;
          }
    }
```

**Πίνακας 11**

## 3.4. Συρραφή βίντεο

### 3.4.1. Κλάση Join

Με την κλάση Join υλοποιείται η διαδικασία της συρραφής βίντεο. Ο κατασκευαστής της Join καλείται από την κλάση Lista με τα παρακάτω ορίσματα: ένα πίνακα τύπου String ο οποίος περιέχει τα ονόματα και την διαδρομή στο δίσκο των αρχείων βίντεο που πρόκειται να συρραφούν και μία μεταβλητή τύπου String η οποία περιέχει το όνομα και την διαδρομή στο δίσκο του βίντεο που θα προκύψει από τη συρραφή. Αρχικά δημιουργείται ένας MediaLocator (αντικείμενο της κλάσης MediaLocator) για κάθε αρχείο (βίντεο) αλλά και για το αρχείο (βίντεο) εξόδου με την χρήση της μεθόδου createMediaLocator(). Τέλος στον κατασκευαστή της κλάσης γίνεται κλήση της μεθόδου doIt().

Η μέθοδος doIt() ελέγχει την ροή της διαδικασίας συρραφής βίντεο. Στην παράγραφο που ακολουθεί θα αναλυθεί η ροή των ενεργειών που γίνονται για την συρραφή βίντεο στην μέθοδο doIt(), ενώ σε επόμενες ενότητες θα παρουσιαστούν οι κλάσεις και οι μέθοδοι οι οποίες χρησιμοποιούνται στην μέθοδο doIt().

Αρχικά στη μέθοδο doIt() δημιουργείται μία μεταβλητή ContentDescriptor (αντικείμενο της κλάσης ContentDescriptor) η οποία παίρνει τιμή την έξοδο της μεθόδου fileExtToCD(). Στη συνέχεια δημιουργείται ένα αντικείμενο - πίνακας της κλάσης ProcInfo. Η κλάση ProcInfo χρησιμοποιείται για να κρατήσει διάφορα χαρακτηριστικά - τιμές από κάθε βίντεο εισόδου. Έτσι για κάθε βίντεο εισόδου δημιουργείται ένας Processor, με τη βοήθεια του Manager και της μεθόδου createProcessor(), ο οποίος και αποθηκεύεται στο αντικείμενο της κλάσης ProcInfo. Μετά γίνεται κλήση της μεθόδου mathcTracks() με ορίσματα το αντικείμενο της κλάσης ProcInfo και τον ContentDescriptor. Επίσης καλείται και η μέθοδος buildTracks() με παράμετρο το αντικείμενο της κλάσης ProcInfo. Στην συνέχεια δημιουργείται ένα αντικείμενο της κλάσης SuperGlueDataSource. Το αντικείμενο αυτό της κλάσης ουσιαστικά αποτελεί το DataSource, το οποίο ο Processor που στη συνέχεια δημιουργείται παίρνει παράμετρο. Ένας ControllerListener τοποθετείται στον Processor που μόλις δημιουργήθηκε. Με τη χρήση της μεθόδου waitForState() ο Processor μεταβαίνει στην κατάσταση configured και τίθεται σε αυτόν ο ContentDescriptor με την χρήση της μεθόδου setContentDescriptor(). Ξανά με την χρήση της μεθόδου waitForState(), ο Processor μεταβαίνει στην κατάσταση realized. Στη συνέχεια δημιουργείται ένα DataSink (αντικείμενο της κλάσης DataSink) με τη

χρήση της μεθόδου createDataSink() στο οποίο και τοποθετείται ένας DataSinkListener. Τέλος ο Processor και το DataSink ξεκινάνε (start()) και καλείται η μέθοδος waitForFileDone(). Αφού η μέθοδος waitForFileDone() διαγνώσει ότι η διαδικασία της συρραφής των αρχείων βίντεο τελείωσε το DataSink τερματίζεται (close()) και καλείται η μέθοδος ok() της κλάσης JoinVideo η οποία με ένα μήνυμα ενημερώνει το χρήστη ότι η διαδικασία της συρραφής ολοκληρώθηκε.

```
public boolean doIt(MediaLocator inML[], MediaLocator outML) {

        ContentDescriptor cd;

        if ((cd = fileExtToCD(outML.getRemainder())) == null) {

        er="Couldn't figure out from the file extension the type of output needed!";
            return false;
        }

        ProcInfo pInfo[] = new ProcInfo[inML.length];

        for (int i = 0; i < inML.length; i++) {
            pInfo[i] = new ProcInfo();
            pInfo[i].ml = inML[i];

            try {
                pInfo[i].p = Manager.createProcessor(inML[i]);
            } catch (Exception e) {
        er="Cannot create a processor from the given url: \n" + e;
                return false;
            }
        }

        if (!matchTracks(pInfo, cd)) {
            er="Failed to match the tracks.\n"+er1;
            return false;
        }

        if (!buildTracks(pInfo)) {
            er="Failed to build processors for the inputs\n"+er1;
            return false;
        }


        SuperGlueDataSource ds = new SuperGlueDataSource(pInfo);

        Processor p;
        try {
            p = Manager.createProcessor(ds);
        } catch (Exception e) {
            er="Failed to create a processor to join the inputs.";
            return false;
        }

        p.addControllerListener(this);
```

```
        if (!waitForState(p, p.Configured)) {
            er="Failed to configure the processor.";
            return false;
        }


        if ((p.setContentDescriptor(cd)) == null) {
            er="Failed to set the output content descriptor on the processor.";
            return false;
        }

        if (!waitForState(p, p.Realized)) {
            er="Failed to realize the processor.";
            return false;
        }


        DataSink dsink;
        if ((dsink = createDataSink(p, outML)) == null) {
            er="Failed to create a DataSink for the given output MediaLocator: " +
outML+"\n"+er1 ;
            return false;
        }

        dsink.addDataSinkListener(this);
        fileDone = false;



        try {
            p.start();
            dsink.start();
        } catch (IOException e) {
            er="IO error during concatenation";

            return false;
        }

        waitForFileDone();


        try {
            dsink.close();
        } catch (Exception e) {}
        p.removeControllerListener(this);

JoinVideo.frr.setVisible(false);
JoinVideo.ok("Join has be done successfully!");

//see output file format
            TrackControl track[];
            try {
                  ps = Manager.createProcessor(outML);
            } catch (Exception e) {
            }
        waitForState(ps,ps.Configured);
  track = ps.getTrackControls();
                    for (int j = 0; j < track.length; j++) {
```

```
                if (track[j].getFormat() instanceof AudioFormat) {

        outAud=track[j].getFormat().toString();
                } else if (track[j].getFormat() instanceof VideoFormat) {
                        outVid=track[j].getFormat().toString();
                }
           }
        return true;
}
```

**Πίνακας 12**

Σε κάθε βήμα (όπως παρουσιάστηκε παραπάνω) της διαδικασίας συρραφής βίντεο, η οποία επιτελείται από τη μέθοδο doIt(), γίνεται έλεγχος για τυχόν προβλήματα που μπορεί να παρουσιαστούν. Αν προκύψει κάποιο πρόβλημα η διαδικασία σταματάει άμεσα και γίνεται κλήση της μεθόδου Err() της κλάσης JoinVideo, η οποία εμφανίζει ένα μήνυμα στο χρήστη με περιεχόμενο το πρόβλημα που προέκυψε.

Στη μέθοδο matchTracks() γίνεται προσπάθεια να βρεθεί ένα κοινό Format, με το οποίο όλα τα βίντεο θα επανακωδικοποιηθούν έτσι ώστε να είναι δυνατό στη συνέχεια να συρραφούν. Η μέθοδος παίρνει παραμέτρους το αντικείμενο της κλάσης ProcInfo και τον ContentDescriptor. Αρχικά για κάθε γίνονται τα εξής: ο Processor κάθε βίντεο, ο οποίος βρίσκεται αποθηκεμένος στο αντικείμενο της κλάσης ProcInfo, μεταβαίνει στην κατάσταση configured με τη χρήση της μεθόδου waitForState(). Με την μέθοδο getTrackControls() αποθηκεύονται τα TrackControls από τον Processor κάθε βίντεο σε ένα πίνακα τύπου TrackControl. Επίσης για κάθε βίντεο δημιουργείται ένα αντικείμενο της κλάσης TrackInfo. Από το TrackControls του κάθε βίντεο και με τη μέθοδο getFormat() γίνεται έλεγχος αν το κάθε Track είναι Video ή Audio και τα αποτελέσματα αποθηκεύονται στο αντικείμενο της κλάσης TrackInfo. Στον Processor του κάθε βίντεο που βρίσκεται στην κλάση ProcInfo τίθεται ο ContentDescriptor που πήρε παράμετρο η μέθοδος matchTracks(). Τέλος για κάθε τύπο Track (Video ή Audio) γίνεται κλήση της μεθόδου tryMatch().

Η μέθοδος tryMatch() παίρνει παραμέτρους ένα αντικείμενο πίνακα της κλάσης ProcInfo και τον τύπο του Track. Ουσιαστικά η tryMatch παίρνει το Format από το TrackControl του πρώτου βίντεο και με τη μέθοδο getSupportedFormats() παίρνει όλα τα υποστηριζόμενα Format. Από τα υποστηριζόμενα format και με την χρήση της μεθόδου tryTranscode(), η tryMatch() βρίσκει το σωστό Format το οποίο και τοποθετεί στο TrackControl του Processor για κάθε βίντεο. Η διαδικασία αυτή γίνεται για κάθε Track (Audio - Video) ξεχωριστά. Έτσι όλα τα βίντεο θα μπορούν να

ενωθούν αφού τα Tracks τους θα έχουν τα ίδια format. Στην περίπτωση που δεν βρεθεί ένα κοινό Format για τα Tracks κάθε βίντεο, η συρραφή δεν μπορεί να πραγματοποιηθεί. Τέλος εξετάζεται αν όλα τα Tracks τύπου Video έχουν τις ίδιες διαστάσεις, αν όχι, τότε στο Format του Track κάθε αρχείου βίντεο εφαρμόζονται οι διαστάσεις του πρώτου βίντεο.

Η μέθοδος tryTranscode() ελέγχει αν το υποστηριζόμενο format το οποίο πήρε παράμετρο, όταν καλέστηκε από την μέθοδο tryMatch(), μπορεί να εφαρμοστεί στο Track (Video ή Audio αντίστοιχα) του κάθε βίντεο. Στην περίπτωση που το Format μπορεί να εφαρμοστεί στο αντίστοιχο Track του κάθε βίντεο τότε επιστρέφει την τιμή true.

Η μέθοδος buildTracks() παίρνει παράμετρο ένα αντικείμενο της κλάσης ProcInfo. Αρχικά στη buildTracks() τίθεται στον Processor κάθε βίντεο, ο ContentDescriptor.RAW μέσω της μεθόδου setContentDescriptor(). Στην συνέχεια ο Processor κάθε βίντεο μεταβαίνει στην κατάσταση realized με τη χρήση της μεθόδου waitForState(). Με την χρήση της μεθόδου getDataOutput() αποθηκεύεται το PushBufferDataSource από τον Processor του κάθε βίντεο στο αντικείμενο της κλάσης ProcInfo στη μεταβλητή τύπου PushBufferDataSource. Επίσης με τη μέθοδο getStreams() αποθηκεύονται τα PushBufferStream από το PushBufferDataSource της κλάσης ProcInfo στην ανάλογη μεταβλητή στο αντικείμενο της κλάσης TrackInfo.

Η μέθοδος waitForState() μαζί με την κλάση StateWaiter χρησιμοποιούνται για την μετάβαση του Processor, τον οποίο παίρνουν παράμετρο, στην κατάσταση την οποία επίσης παίρνουν παράμετρο. Η μέθοδος waitForState() δημιουργεί ένα αντικείμενο της κλάσης StateWaiter και καλεί την μέθοδο waitForState() της κλάσης StateWaiter. Η μέθοδος waitForState() του αντικειμένου της κλάσης StateWaiter σταματάει την ροή της διαδικασίας της συρραφής, έως ότου ο Processor περάσει στην κατάσταση που του δόθηκε. Αφού ο Processor μεταβεί χωρίς προβλήματα στην κατάσταση που του δόθηκε, επιστρέφεται η τιμή true.

Η μέθοδος createDataSink παίρνει το DataSource από τον Processor με την χρήση της μεθόδου getDataOutput(). Στη συνέχεια δημιουργεί ένα DataSink (αντικείμενο της κλάσης DataSink) με το DataSource που πήρε από τον Processor και τον MediaLocator του αρχείου εξόδου. Τέλος η μέθοδος επιστρέφει το DataSink που δημιούργησε.

Η μέθοδος waitForFileDone() σταματάει προσωρινά την ροή του προγράμματος (δηλαδή το πρόγραμμα δεν θα μπορεί να χρησιμοποιηθεί) μέχρι να

ολοκληρωθεί η διαδικασία της συρραφής, δηλαδή μέχρι να ολοκληρωθεί η εγγραφή του αρχείου εξόδου στο δίσκο.

Η μέθοδος fileExtToCD() παίρνει παράμετρο μία μεταβλητή τύπου String η οποία περιέχει ο όνομα και τη διαδρομή στο δίσκο του αρχείου εξόδου. Στη συνέχεια απομονώνει την κατάληξη του αρχείου (π.χ. .avi) και επιστρέφει τον ContentDescriptor που παίρνει από την κατάληξη του αρχείου με την χρήση της μεθόδου getMimeType() του MimeManager.

### *3.4.2. Κλάσεις ProcInfo και TrackInfo*

Οι κλάσεις ProcInfo και TrackInfo περιέχουν διάφορες μεταβλητές στις οποίες αποθηκεύονται χαρακτηριστικά των βίντεο εισόδου. Κάθε αντικείμενο της κλάσης ProcInfo αντιπροσωπεύει ένα βίντεο εισόδου, ενώ κάθε αντικείμενο της κλάσης TrackInfo αντιπροσωπεύει ένα Track (Video ή Audio) από ένα βίντεο εισόδου. Οπότε για κάθε αντικείμενο της κλάσης ProcInfo δημιουργούνται δύο αντικείμενα της κλάσης TrackInfo, ένα για το Track – Video και ένα για το Track – Audio.

Η κλάση ProcInfo περιέχει μία μεταβλητή τύπου MediaLocator η οποία παίρνει τιμή τον MediaLocator του εκάστοτε βίντεο, μία μεταβλητή τύπου Processor στην οποία αποθηκεύεται ο Processor που δημιουργείται για κάθε βίντεο, μία μεταβλητή τύπου PushBufferDataSource στην οποία αποθηκεύεται το περιεχόμενο του DataSource του Processor κάθε βίντεο και ένας πίνακας αντικειμένων της κλάσης TrackInfo.

Η κλάση TrackInfo περιέχει μία μεταβλητή τύπου TrackControl στην οποία αποθηκεύεται το TrackControl του αντίστοιχου Track, μία μεταβλητή τύπου PushBufferStream η οποία παίρνει το Stream του αντίστοιχου Track, από το DataSource, μία μεταβλητή τύπου int η οποία υποδηλώνει το Track από τα δύο που έχει το βίντεο εισόδου(το πρώτο ή το δεύτερο) και μία μεταβλητή τύπου Boolean η οποία χρησιμοποιείται για να δείξει ότι το αντίστοιχο Track έχει χρησιμοποιηθεί στη συρραφή των βίντεο.

```
public class TrackInfo {
    public TrackControl tc;
    public PushBufferStream pbs;
    public int idx;
    public boolean done;
```

```
        public boolean disabled;
    }
    public class ProcInfo {
        public MediaLocator ml;
        public Processor p;
        public PushBufferDataSource ds;
        public TrackInfo tracksByType[];
        public int numTracksByType[];
        public int numTracks;
    }
```

**Πίνακας 13**

### *3.4.3. Κλάση SuperGlueDataSource*

Η κλάση SuperGlueDataSource αποτελεί ουσιαστικά το DataSource το οποίο χρησιμοποιεί ο Processor για να προχωρήσει στη συρραφή των βίντεο. Αρχικά στην κλάση δημιουργείται ένα αντικείμενο – πίνακας της κλάσης SuperGlueStream με πεδία όσα και τα βίντεο που πρόκειται να συρραφούν. Τέλος γίνεται κλήση της μεθόδου, setStreams() για το πρώτο βίντεο. Στη μέθοδο setStreams() γίνεται κλήση της μεθόδου setStream() της κλάσης SuperGlueStream για κάθε Track του βίντεο. Επίσης πριν γίνει κλήση της setStream() γίνεται έλεγχος αν πρόκειται για Track Audio και αν αυτό(το Track) είναι κωδικοποιημένο σε RAW μορφή, έτσι ώστε να γίνει στη συνέχεια ευκολότερα ο υπολογισμός της διάρκειας του βίντεο. Στη συνέχεια στη μέθοδο start() (η μέθοδος start() ανήκει στην PushBufferDataSource, υπερκλάση της SuperGlueDataSource) ξεκινάνε (start()) ο Processor και το PushBufferDataSource του αντίστοιχου βίντεο, που είναι αποθηκεμένα στην ProcInfo, αφού από εκεί θα πάρει τα δεδομένα ο Processor για κάθε αρχείο. Η μέθοδος getStreams() επιστρέφει τα Streams τα οποία δημιουργούνται στην κλάση SuperGlueStream και είναι τύπου PushBufferStream, τα οποία και χρησιμοποιούνται στο DataSink.

Η μέθοδος handleEOM καλείται από την κλάση SuperGlueStream για να συνεχιστεί η ροή της διαδικασίας συρραφής, έως ότου διαβαστούν όλα τα Track από όλα τα βίντεο. Η handleEOM αρχικά ελέγχει αν όλα τα Track του βίντεο έχουν διαβαστεί. Αν δεν έχουν διαβαστεί επιστρέφει false για να διαβαστεί το επόμενο Track από το ίδιο βίντεο, αλλιώς αν έχουν διαβαστεί όλα τα Tracks του βίντεο, σταματάει (stop()) τον Processor και το PushBufferDataSource του βίντεο. Στη συνέχεια ελέγχει αν το βίντεο που διαβάστηκε τελευταίο, είναι το τελευταίο βίντεο από αυτά που πρόκειται να συρραφούν. Αν δεν είναι το τελευταίο βίντεο καλεί τη μέθοδο setStreams() για το επόμενο βίντεο.

Στην κλάση SuperGlueDataSource, τέλος, εφαρμόζονται και οι υπόλοιπες μέθοδοι τις οποίες η κλάση κληρονομεί από την υπερκλάση της PushBufferDataSource.

```
class SuperGlueDataSource extends PushBufferDataSource {

        ProcInfo pInfo[];
        int current;
        SuperGlueStream streams[];

        public SuperGlueDataSource(ProcInfo pInfo[]) {
           this.pInfo = pInfo;
           streams = new SuperGlueStream[2];
           for (int i = 0; i < 2; i++)
                 streams[i] = new SuperGlueStream(this);
           current = 0;
           setStreams(pInfo[current]);
        }

        void setStreams(ProcInfo pInfo) {
           int j = 0;
           masterFound = false;
           for (int type = AUDIO; type < MEDIA_TYPES; type++) {

                    if (!masterFound &&
                          isRawAudio(pInfo.tracksByType[type])) {
                          streams[j].setStream(pInfo.tracksByType[type], true);
                          masterFound = true;
                    } else
                          streams[j].setStream(pInfo.tracksByType[type], false);
                    j++;
           }
        }

        public void connect() throws java.io.IOException {
        }

        public PushBufferStream [] getStreams() {
           return streams;
        }

        public void start() throws java.io.IOException {
           pInfo[current].p.start();
           pInfo[current].ds.start();
        }

        public void stop() throws java.io.IOException {
        }

        synchronized boolean handleEOM(TrackInfo tInfo) {
           boolean lastProcessor = (current >= pInfo.length - 1);


           for (int type = AUDIO; type < MEDIA_TYPES; type++) {

                    if (!pInfo[current].tracksByType[type].done)
```

```
                        return lastProcessor;
            }

            try {
                pInfo[current].p.stop();
                pInfo[current].ds.stop();
            } catch (Exception e) {}

            if (lastProcessor) {

                return lastProcessor;
            }

            if (!masterFound &&
                pInfo[current].p.getDuration() != Duration.DURATION_UNKNOWN) {
                masterTime += pInfo[current].p.getDuration().getNanoseconds();
            }

            current++;
            setStreams(pInfo[current]);
            try {
                start();
            } catch (Exception e) {}
            return lastProcessor;
        }

        public Object getControl(String name) {
            return null;
        }

        public Object [] getControls() {
            return new Control[0];
        }

        public Time getDuration() {
            return Duration.DURATION_UNKNOWN;
        }

        public void disconnect() {
        }

        public String getContentType() {
            return ContentDescriptor.RAW;
        }

        public MediaLocator getLocator() {
            return pInfo[current].ml;
        }

        public void setLocator(MediaLocator ml) {
        }
    }
```

**Πίνακας 14**

### *3.4.4. Κλάση SuperGlueStream*

Η κλάση SuperGlueStream αντιπροσωπεύει το σύνολο των Streams τα οποία διαχειρίζεται το DataSource και χρησιμοποιεί ο Processor. Η SuperGlueStream μέσω της μεθόδου read() διαβάζει από το PushBufferStream του κάθε Track από κάθε βίντεο τα δεδομένα και τα βάζει σε Buffer. Τα Buffer αυτά αποτελούν τα Streams του DataSource. Επίσης μέσω των μεθόδων getTimeStamp() και setTimeStamp υπολογίζεται ο χρόνος κάθε Track και τίθεται στο Buffer. Στην περίπτωση που το Track είναι Audio και είναι κωδικοποιημένο σε RAW μορφή, τότε με τις μεθόδους getLength() και computeDuration() υπολογίζεται η διάρκεια του Buffer μέχρι και το συγκεκριμένο Track. Όταν πρόκειται για Track από το ίδιο βίντεο, δηλαδή όταν το ένα Track έχει περάσει στο Buffer, στο δεύτερο Track τίθεται ο ίδιος χρόνος στο Buffer μέσω της μεθόδου setTimeStamp() έτσι ώστε τα Track του ίδιου βίντεο να είναι συγχρονισμένα στον ίδιο χρόνο. Στη συνέχεια ελέγχεται αν στο Buffer έχουν περάσει όλα τα δεδομένα από το Track και γίνεται κλήση της μεθόδου handleEOM() της κλάσης SuperGlueDataSource, όπου ανάλογα είτε διαβάζεται το επόμενο Track του βίντεο είτε διαβάζονται τα Tracks του επόμενου βίντεο. Τέλος εφαρμόζονται οι μέθοδοι τις οποίες η κλάση κληρονομεί (implements) από τις διασυνδέσεις (interfaces) PushBufferStream και BufferTransferHandler.

```
class SuperGlueStream implements PushBufferStream, BufferTransferHandler {

        SuperGlueDataSource ds;
        TrackInfo tInfo;
        PushBufferStream pbs;
        BufferTransferHandler bth;
        boolean useAsMaster = false;
        long timeStamp = 0;
        long lastTS = 0;

        public SuperGlueStream(SuperGlueDataSource ds) {
            this.ds = ds;
        }

        public void setStream(TrackInfo tInfo, boolean useAsMaster) {
            this.tInfo = tInfo;
            this.useAsMaster = useAsMaster;
            if (pbs != null)
                pbs.setTransferHandler(null);
            pbs = tInfo.pbs;

            if (masterTime > 0)
                timeStamp = masterTime;
            lastTS = 0;

            pbs.setTransferHandler(this);
```

```
        }

        public void read(Buffer buffer) throws IOException {
            pbs.read(buffer);


            if (buffer.getTimeStamp() != Buffer.TIME_UNKNOWN) {
                long diff = buffer.getTimeStamp() - lastTS;
                lastTS = buffer.getTimeStamp();
                if (diff > 0)
                    timeStamp += diff;
                buffer.setTimeStamp(timeStamp);
            }

            if (useAsMaster) {
                if (buffer.getFormat() instanceof AudioFormat) {
                    AudioFormat af = (AudioFormat)buffer.getFormat();
                    masterAudioLen += buffer.getLength();
                    long t = af.computeDuration(masterAudioLen);
                    if (t > 0) {
                        masterTime = t;
                    } else {
                        masterTime = buffer.getTimeStamp();
                    }
                } else {
                    masterTime = buffer.getTimeStamp();
                }
            }

            if (buffer.isEOM()) {
                tInfo.done = true;
                if (!ds.handleEOM(tInfo)) {
                    buffer.setEOM(false);
                    buffer.setDiscard(true);
                }
            }
        }

        public ContentDescriptor getContentDescriptor() {
            return new ContentDescriptor(ContentDescriptor.RAW);
        }

        public boolean endOfStream() {
            return false;
        }

        public long getContentLength() {
            return LENGTH_UNKNOWN;
        }

        public Format getFormat() {
            return tInfo.tc.getFormat();
        }

        public void setTransferHandler(BufferTransferHandler bth) {
            this.bth = bth;
        }

        public Object getControl(String name) {
```

```
            return null;
        }

        public Object [] getControls() {

            return new Control[0];
        }

        public synchronized void transferData(PushBufferStream pbs) {
          if (bth != null)
                bth.transferData(this);
        }
    }
```

**Πίνακας 15**

# 4. Οδηγίες χρήσης της εφαρμογής

Στην ενότητα που ακολουθεί γίνεται πλήρης αναφορά του τρόπου λειτουργίας της εφαρμογής με τις αναλυτικές οδηγίες χρήσης που παρατίθενται. Αρχικά γίνεται παρουσίαση των περιεχομένων της επιφάνειας εργασίας της εφαρμογής και μετά γίνεται ανάλυση των γενικών λειτουργιών που η εφαρμογή προσφέρει. Στη συνέχεια περιγράφεται ο τρόπος με τον οποίο γίνεται η επιλογή των αρχείων βίντεο ή εικόνων και η όλη διαδικασία που ακολουθεί μέχρι και τη συρραφή αυτών.

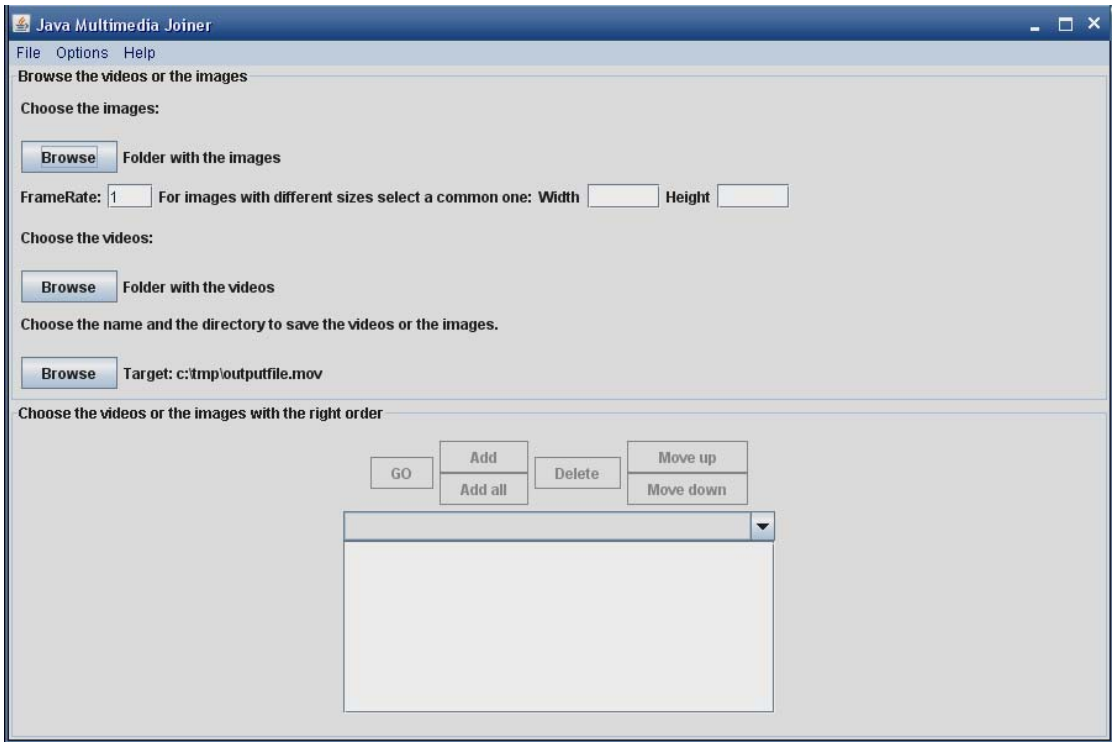
## 4.1. Γενική περιγραφή

Για να ξεκινήσει η εφαρμογή ο χρήστης πρέπει να καλέσει μέσω της Java την κλάση JoinVideo. Η κλήση γίνεται μέσα από το φάκελο στον οποίο βρίσκεται η κλάση JoinVideo με την εντολή "java JoinVideo" στο Command Prompt (Command Line) των Windows. Για το σκοπό αυτό έχει δημιουργηθεί το αρχείο "Join.bat" (script file) το οποίο αφού εκτελεστεί, τρέχει αυτόματα την εντολή "java JoinVideo" στο Command Prompt.

Αφού γίνει λοιπόν κλήση της κλάσης JoinVideo εμφανίζεται η επιφάνεια εργασίας της εφαρμογής. Στο πάνω μέρος της εφαρμογής υπάρχει το βασικό μενού επιλογών. Στο μενού επιλογών ο χρήστης μπορεί να επιλέξει από το μενού "File", την επιλογή "New" για να φέρει την εφαρμογή στην αρχική κατάσταση (να εξαφανίσει δηλαδή της επιλογές που είχε κάνει προηγουμένως), την επιλογή "Open Image" για να εμφανίσει μία εικόνα, την επιλογή "Open Video" για να αναπαραγάγει ένα βίντεο και την επιλογή "Exit" για να κλείσει την εφαρμογή. Στο μενού "Options", με την επιλογή "Codecs" ο χρήστης μπορεί να δει τα βασικά χαρακτηριστικά των Format των βίντεο εισόδου και του βίντεο εξόδου αφού γίνει η συρραφή και με την επιλογή "Play output file" ο χρήστης μπορεί να αναπαραγάγει το αρχείο (βίντεο) εξόδου. Τέλος στο μενού "Help" και με την επιλογή "About" ο χρήστης μπορεί να δει κάποιες βασικές πληροφορίες για την εφαρμογή.

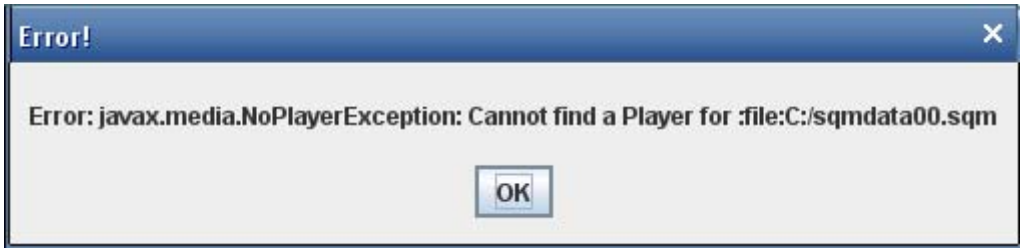
Το κύριο μέρος της εφαρμογής αποτελείται από δύο τμήματα (δύο πλαίσια τα οποία έχουν και τους αντίστοιχους τίτλους). Στο πρώτο τμήμα δίνεται η δυνατότητα στο χρήστη να επιλέξει τα αρχεία που θέλει να συρράψει και το αρχείο στο οποίο θα αποθηκευτεί το βίντεο της συρραφής και να δώσει κάποια βασικά χαρακτηριστικά, στην περίπτωση συρραφής εικόνων, για το αρχείο εξόδου.

Στο δεύτερο τμήμα ο χρήστης έχει την δυνατότητα να διαχειριστεί τα αρχεία που έχει επιλέξει και στη συνέχεια να προχωρήσει στη συρραφή. Γενικότερα ο χρήστης μπορεί να αλλάξει την σειρά των αρχείων που έχει επιλέξει, να αφαιρέσει προσθέσει μερικά από αυτά και στη συνέχεια να προχωρήσει στη συρραφή.




**Εικόνα 1**

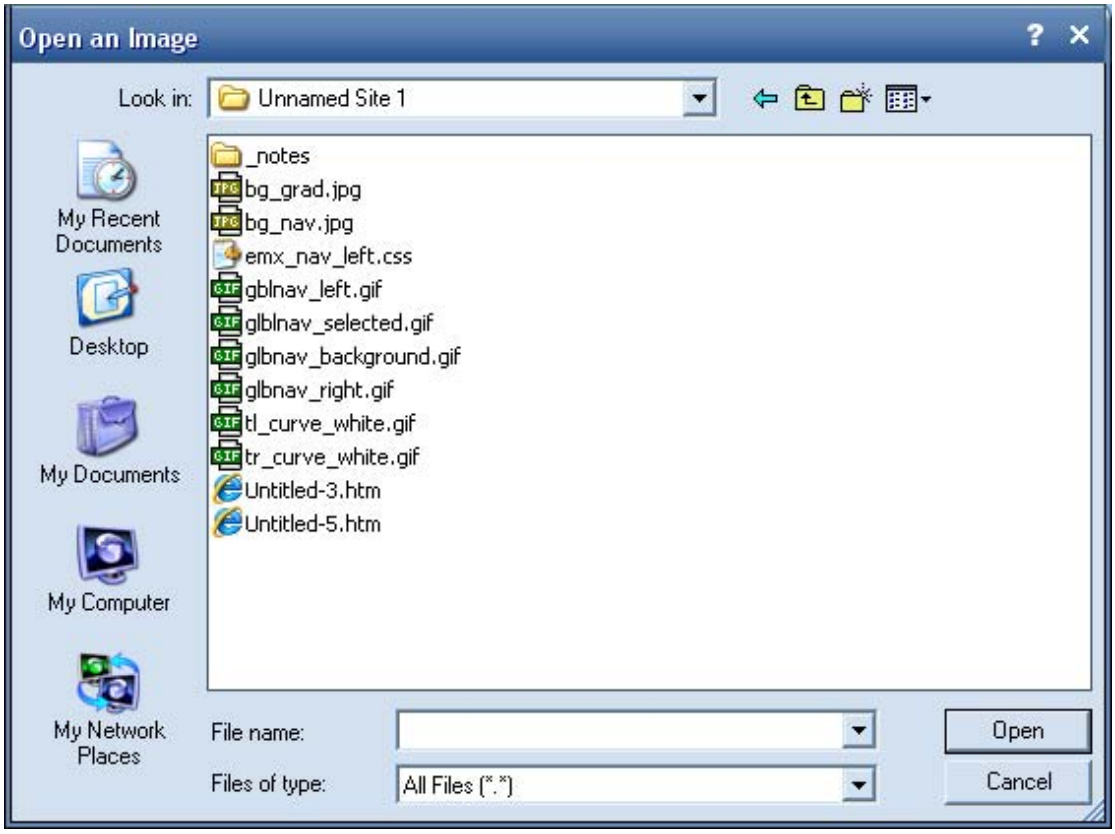
Στην περίπτωση που ο χρήστης έχει επιλέξει να κάνει κάποια λειτουργία στην εφαρμογή και κατά την διάρκεια που αυτή επιτελείται δημιουργηθεί κάποιο πρόβλημα, τότε εμφανίζεται στο χρήστη ένα μήνυμα το οποίο περιέχει τους λόγους για τους οποίους δημιουργήθηκε το πρόβλημα αυτό.



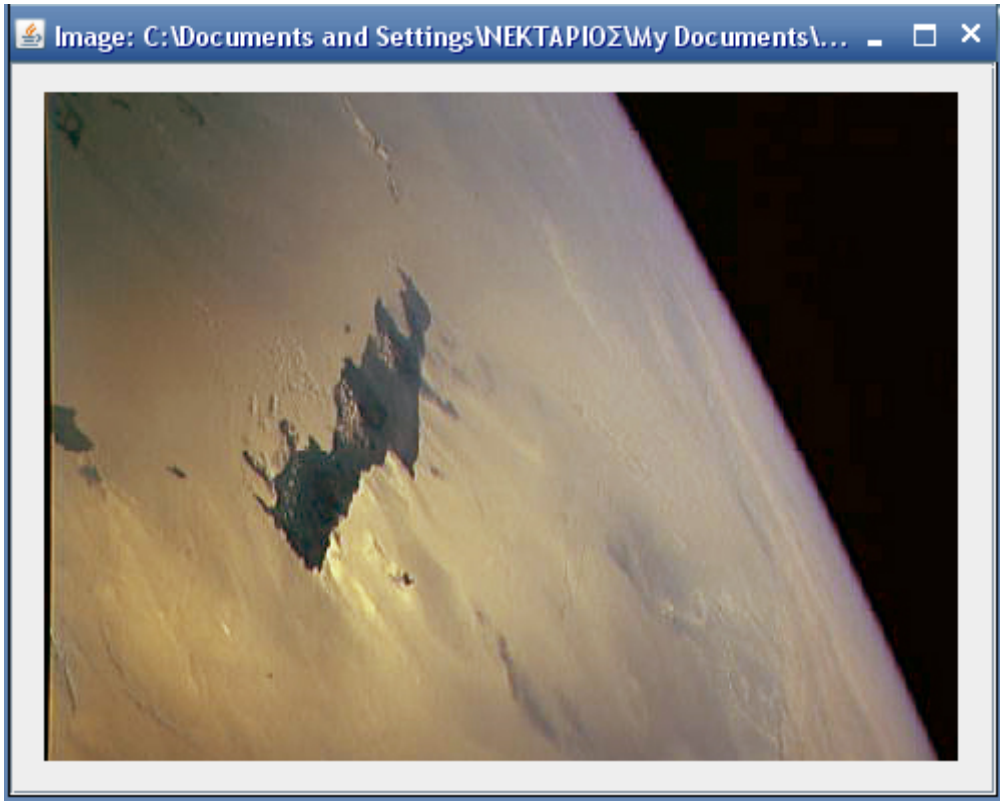

**Εικόνα 2**

## 4.2. Γενικές λειτουργίες

Όπως και παραπάνω ειπώθηκε, από την επιλογή “Open Image” του μενού “File” ο χρήστης μπορεί να εμφανίσει μία εικόνα. Η επιλογή αυτή ανοίγει ένα παράθυρο διαλόγου στο οποίο ο χρήστης μπορεί να αναζητήσει και να επιλέξει στην συνέχεια την εικόνα που επιθυμεί να εμφανίσει. Αφού ο χρήστης επιλέξει την εικόνα που επιθυμεί να εμφανίσει, αυτή παρουσιάζεται σε ένα νέο παράθυρο με τίτλο το όνομα της εικόνας και την διαδρομή αυτής στο δίσκο.




**Εικόνα 3**




**Εικόνα 4**

Παρόμοια από την επιλογή “Open Video” του μενού “File” ο χρήστης μπορεί να αναπαραγάγει ένα βίντεο. Η επιλογή αυτή ανοίγει ένα παράθυρο διαλόγου στο οποίο ο χρήστης μπορεί να αναζητήσει και να επιλέξει στη συνέχεια το βίντεο που επιθυμεί να αναπαραγάγει. Η αναπαραγωγή του βίντεο γίνεται σε ένα νέο παράθυρο με τίτλο το όνομα και τη διαδρομή στο δίσκο του βίντεο. Στο παράθυρο αυτό παρέχονται στο χρήστη οι βασικές λειτουργίες ελέγχου κατά την αναπαραγωγή του βίντεο. Για παράδειγμα ο χρήστης μπορεί να σταματήσει την αναπαραγωγή του βίντεο, να προχωρήσει τη ροή αναπαραγωγής του βίντεο μέσω του δείκτη στην ράβδο κύλισης και να αυξήσει ή να μειώσει την ένταση του ήχου.

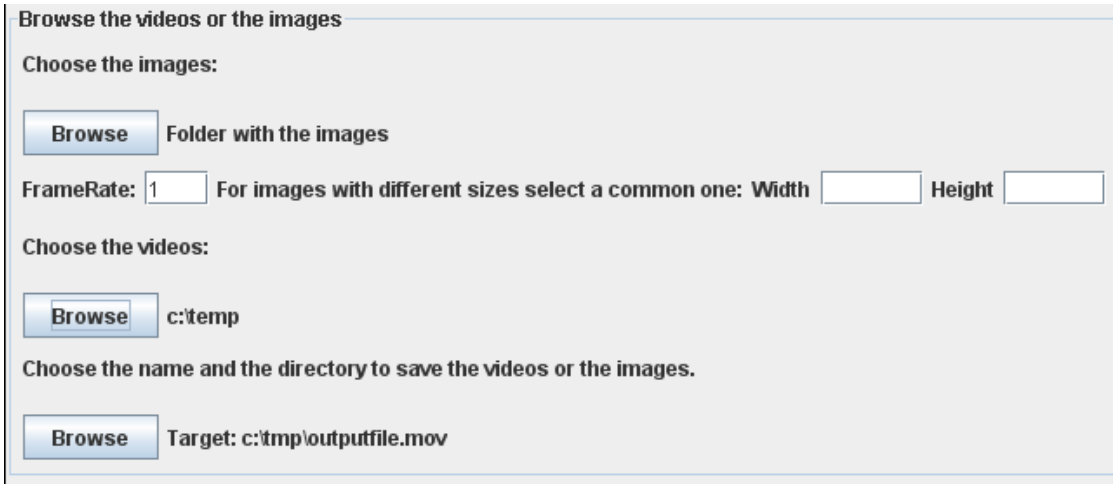

**Εικόνα 5**

Η επιλογή "Video Codecs" του μενού "Options" ανοίγει ένα παράθυρο στο οποίο εμφανίζονται τα βασικά χαρακτηριστικά των Format των βίντεο που χρησιμοποιήθηκαν στην συρραφή αλλά και του βίντεο εξόδου. Τα χαρακτηριστικά αυτά μπορούν να εμφανιστούν μόνο μετά την συρραφή των βίντεο.


**Εικόνα 6**

## 4.3. Επιλογή και συρραφή των αρχείων

Στο πρώτο μέρος της επιφάνειας εργασίας της εφαρμογής υπάρχουν τρία κουμπιά με τον τίτλο "Browse". Το πρώτο κουμπί ανοίγει ένα παράθυρο διαλόγου στο οποίο ο χρήστης επιλέγει τις εικόνες τις οποίες επιθυμεί να συρράψει. Στο παράθυρο αυτό ο χρήστης έχει την δυνατότητα να επιλέξει μία εικόνα κάθε φορά ή ένα ολόκληρο φάκελο με εικόνες. Οι εικόνες που επιλέγει ο χρήστης προστίθενται στο πτυσσόμενο πλαίσιο που βρίσκεται παρακάτω.




**Εικόνα 7**

Παρόμοια το δεύτερο κουμπί ανοίγει ένα παράθυρο διαλόγου στο οποίο ο χρήστης επιλέγει τα βίντεο τα οποία επιθυμεί να συρράψει. Στο παράθυρο αυτό ο χρήστης έχει την δυνατότητα να επιλέξει ένα βίντεο κάθε φορά ή ένα ολόκληρο φάκελο με βίντεο. Τα βίντεο που επιλέγει ο χρήστης προστίθενται στο πτυσσόμενο πλαίσιο που βρίσκεται παρακάτω.

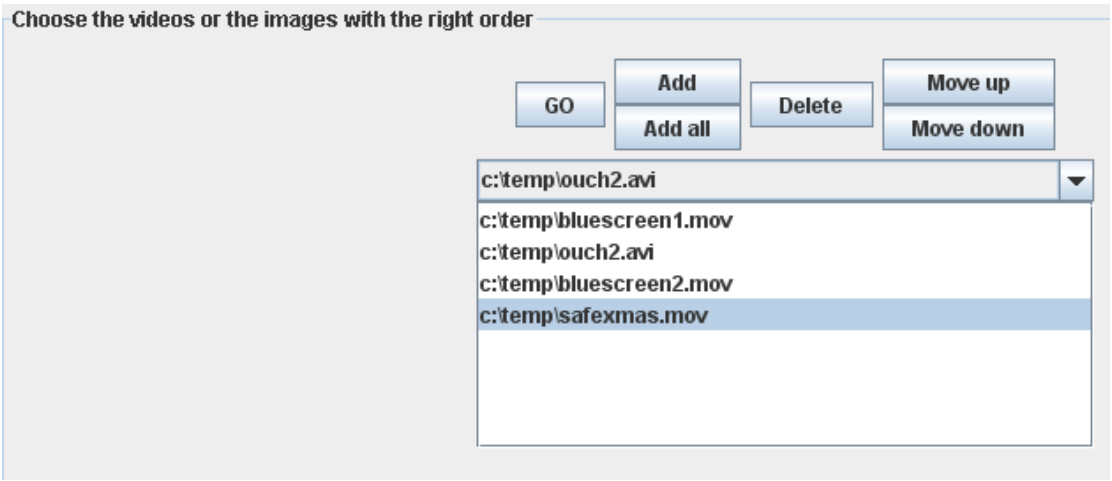
Επειδή στην εφαρμογή δεν υπάρχει η δυνατότητα της συρραφής βίντεο και εικόνων μαζί, όταν για παράδειγμα ο χρήστης έχει επιλέξει εικόνες για συρραφή και στη συνέχεια προσπαθήσει να επιλέξει βίντεο για να τα προσθέσει στο πτυσσόμενο πλαίσιο, τότε αφαιρούνται οι εικόνες που είχε επιλέξει ο χρήστης και προστίθενται μόνο τα βίντεο.

Το τρίτο κουμπί με τίτλο "Browse" ανοίγει ένα παράθυρο διαλόγου στο οποίο γίνεται η επιλογή του αρχείου στο οποίο θα αποθηκευτεί το αποτέλεσμα της συρραφής. Πολύ σημαντικό είναι ο χρήστης να γράψει σωστά την κατάληξη (π.χ. .avi) του αρχείου γιατί από την κατάληξη αυτή στη συνέχεια η εφαρμογή θα

συμπεράνει τον τύπο του βίντεο που θα αποθηκεύσει. Η Java υποστηρίζει τους τύπους αρχείων βίντεο MOV και AVI.

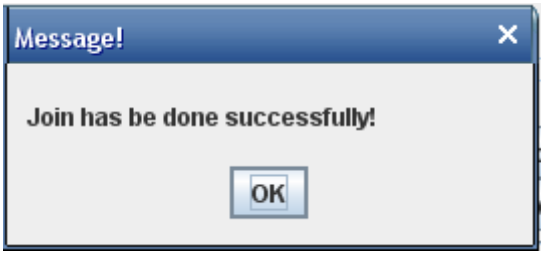
Επίσης κάτω από το κουμπί "Browse" στο οποίο επιλέγονται εικόνες για συρραφή, υπάρχουν τρία πεδία κειμένου. Το πρώτο πεδίο κειμένου με τίτλο "FrameRate" υποδηλώνει τον αριθμό των εικόνων που θα εμφανίζονται στο βίντεο σε διάστημα ενός δευτερολέπτου. Το πεδίο αυτό μπορεί να πάρει και δεκαδική τιμή. Δηλαδή αν ο χρήστης δώσει την τιμή 0.5 το βίντεο θα εμφανίζει μία εικόνα κάθε δύο δευτερόλεπτα. Τα άλλα δύο πεδία κειμένου με τίτλους "Width" και "Height" υποδηλώνουν τις διαστάσεις που θα έχει το βίντεο εξόδου. Αν τα πεδία αυτά μείνουν κενά τότε το βίντεο θα πάρει της διαστάσεις της πρώτης εικόνας.

Στο δεύτερο πλαίσιο της επιφάνειας εργασίας της εφαρμογής έχουμε πέντε κουμπιά, μία πτυσσόμενη λίστα και μία λίστα. Στην πτυσσόμενη λίστα υπάρχουν τα βίντεο που έχει επιλέξει ο χρήστης. Με τα κουμπιά "Add" και "Add all" ο χρήστης τοποθετεί στη λίστα το αρχείο που έχει επιλέξει ή όλα τα αρχεία από την πτυσσόμενη λίστα αντίστοιχα. Επίσης με τα κουμπιά "Move up" και "Move down" ο χρήστης μπορεί να αλλάξει τη σειρά των αρχείων στη λίστα. Αφού ο χρήστης τελειώσει με την επιλογή και διαχείριση των αρχείων με το κουμπί "GO" μπορεί να ξεκινήσει την διαδικασία της συρραφής.




**Εικόνα 8**

Τέλος αφού η συρραφή των αρχείων τελειώσει με επιτυχία, εμφανίζεται στο χρήστη ένα μήνυμα για να τον ενημερώσει. Με την επιλογή "Play output file" στο μενού "Options" ο χρήστης μπορεί να αναπαραγάγει το βίντεο που δημιούργησε με την συρραφή των αρχείων.




**Εικόνα 9**

# 5. Συμπεράσματα

## 5.1. Ανάπτυξη της εφαρμογής

Η εφαρμογή μπορεί να αναπτυχθεί σε διάφορους τομείς έτσι ώστε να συμπεριλάβει περισσότερες δυνατότητες. Θα είναι πολύ χρήσιμο, για τον χρήστη της εφαρμογής, να υπάρχει η δυνατότητα επιλογής του Format για το παραγόμενο βίντεο. Επίσης μπορεί να προστεθεί η δυνατότητα να χρησιμοποιηθούν τα βασικά εφφέ μετάβασης κατά την συρραφή βίντεο όπως και η δυνατότητα συρραφής αρχείων εικόνων και αρχείων βίντεο μαζί.

## 5.2. Περιορισμοί / Τεχνικές δυσκολίες

Κατά την υλοποίηση της εφαρμογής που παρουσιάστηκε εντοπίστηκαν διάφορα προβλήματα και περιορισμοί στον τρόπο με τον οποίο η Java επικοινωνεί με τα αρχεία πολυμέσων. Κρίθηκε σκόπιμο, τα προβλήματα και οι περιορισμοί που εντοπίστηκαν, να παρουσιαστούν παρακάτω.

### *Κλάση ImageToBuffer*

Το Buffer το οποίο δίνει η κλάση ImageToBuffer κατά την διάρκεια της συρραφής εικόνων έχει υποχρεωτικά Format RGB. Έτσι αφού δεν υπάρχει δυνατότητα αλλαγής του Format στο Buffer το βίντεο που θα προκύψει από την συρραφή θα έχει Format RGB.

### *Multiplexer*

Ο Multiplexer που υλοποιείται με τον Processor και χρησιμοποιείται για να δημιουργηθεί το βίντεο εξόδου υποστηρίζει τους τύπους “MOV” και “AVI” για τα αρχεία εξόδου. Αυτό σημαίνει ότι τα αρχεία που θα δημιουργηθούν από την συρραφή βίντεο ή εικόνων θα μπορούν να είναι μόνο τύπου “MOV” και “AVI”.

### *Format Video – Audio*

Η Java εμφανίζει πολλά προβλήματα στην διαχείριση των Format με αποτέλεσμα να δημιουργούνται έτσι πολλά “Exceptions”. Χρειάστηκε να γίνουν πολλές δοκιμές έτσι ώστε να βρεθούν τα σημεία όπου η Java δημιουργούσε “Exceptions” κατά την διαχείριση των Format. Στα σημεία αυτά ρυθμίστηκε κατάλληλα ο κώδικας για να μην διακόπτεται βίαια (κολλάει) η λειτουργία της εφαρμογής.

# *Βιβλιογραφία*


- Εισαγωγή στη Java 2, Γιώργος Λιακέας, εκδ. Κλειδάριθμος
- http://java.sun.com/products/java-media/jmf/
- http://en.wikipedia.org/wiki/Java_Media_Framework
- http://www.javaworld.com/jw-04-2001/jw-0406-jmf1.html
- http://java.sun.com/developer/onlineTraining/index.html
- http://java.sun.com/products/java-media/jmf/1.0/guide/index.html
- http://java.sun.com/docs/books/tutorial/

# *Παράρτημα*

## *Κώδικας κλάσης JoinVideo*

```
import com.sun.media.ui.*;
import javax.media.*;
import javax.media.protocol.*;
import javax.media.protocol.DataSource;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.filechooser.*;
import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.Vector;
import java.awt.image.*;
import java.awt.image.BufferedImage;

public class JoinVideo extends JFrame{

   public static void main(String args[]) {
           JoinVideo video = new JoinVideo();
   }

   public static void ok(String s) {
    JOptionPane.showMessageDialog
           (null,s,"Message!",JOptionPane.PLAIN_MESSAGE);
 }
     public static void Err(String s) {
    JOptionPane.showMessageDialog
           (null,s,"Error!",JOptionPane.PLAIN_MESSAGE);
 }
   JMFrame jmframe=null;
   public static JFrame frr;
   JPanel pan;
   JLayeredPane pan1,pan2;
   String type;
   FileDialog fd=null;
   FileDialog fd1=null;
   JFileChooser fc=null;
   JFileChooser fc1=null;
   JFileChooser fc2=null;
   Player player = null;
   Player newPlayer = null;
   String filename;
   JButton browse;
                   JButton browse1;
                   JButton browse2;
   JLabel path, path1;
                   JLabel c;
                   String s;
   File sav;
   File direct,direct1;
   File [] arxeio,arxeio1;
```

```
   File [] files,files1;
   Lista lista;
                    private static File currentDirectory=null;
          private static File currentDirectory1=null;
          public static JTextField width,height,frameRate;

   public JoinVideo() {
super("Java Multimedia Joiner");

   browse=new JButton("Browse");
   browse1=new JButton("Browse");
   browse2=new JButton("Browse");
   JLabel ef=new JLabel("Choose the name and the directory to save the videos or
the images.");
   JLabel b=new JLabel("Choose the videos:");
   JLabel d=new JLabel("Choose the images:");
   JLabel different=new JLabel("For images with different sizes select a common
one:");
   JLabel fr=new JLabel("FrameRate:");
   width=new JTextField(5);
   height=new JTextField(5);
   frameRate=new JTextField("1",3);
   c=new JLabel("Target: c:\\tmp\\outputfile.mov");
   s="file:c:\\tmp\\outputfile.mov";
   path=new JLabel("Folder with the videos                                    ");
   path1=new JLabel("Folder with the images                                     ");

         // Add the desktop pane
   pan1 = new JLayeredPane();
   pan1.setPreferredSize(new Dimension(250, 150));
   pan1.setBorder(BorderFactory.createTitledBorder("Browse the videos or the
images"));
   pan2 = new JLayeredPane();
   pan2.setBorder(BorderFactory.createTitledBorder("Choose the videos or the
images with the right order"));
            pan1.setLayout(new GridLayout(7,1));
            pan2.setLayout(new FlowLayout());
            pan1.setOpaque(false);
   pan2.setOpaque(false);

            JPanel miniPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
miniPanel.setOpaque(false);
            miniPanel.add(d);
            pan1.add(miniPanel);
            miniPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
miniPanel.setOpaque(false);
            miniPanel.add(browse2);
            miniPanel.add(path1);
            pan1.add(miniPanel);
            miniPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
miniPanel.setOpaque(false);
            miniPanel.add(fr);
            miniPanel.add(frameRate);
            miniPanel.add(different);
            miniPanel.add(new JLabel("Width"));
            miniPanel.add(width);
            miniPanel.add(new JLabel("Height"));
            miniPanel.add(height);
            pan1.add(miniPanel);
```

```
            miniPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
miniPanel.setOpaque(false);
            miniPanel.add(b);
            pan1.add(miniPanel);
            miniPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
miniPanel.setOpaque(false);
            miniPanel.add(browse);
            miniPanel.add(path);
            pan1.add(miniPanel);
            miniPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
miniPanel.setOpaque(false);
            miniPanel.add(ef);
            pan1.add(miniPanel);
            miniPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
miniPanel.setOpaque(false);
            miniPanel.add(browse1);
            miniPanel.add(c);
            pan1.add(miniPanel);

            lista=new Lista();
    pan2.add(lista);

  pan=new JPanel();
  pan.setLayout(new GridLayout(2,1));
  pan.setOpaque(false);
  pan.add(pan1);
  pan.add(pan2);
  getContentPane().add(pan);

  setMenuBar(createMenuBar());
          setSize(900,600);
          setVisible(true);

          addWindowListener( new WindowAdapter() {
             public void windowClosing(WindowEvent we) {
                   System.exit(0);
             }
          } );

//Wait frame
          frr=new JFrame("Please wait!");
          frr.setSize(200,45);
          frr.setLocation(200, 200);
          frr.setResizable(false);
  frr.setVisible(false);

browse.addActionListener(new ActionListener(){
          public void actionPerformed(ActionEvent e){
          if(currentDirectory==null){
                   fc=new JFileChooser("c:/");
          }else {
                   fc=new JFileChooser(currentDirectory);
          }
            fc.setFileHidingEnabled(true);
                   fc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
        fc.setMultiSelectionEnabled(false);
        fc.setDragEnabled(true);

             int returnVal = fc.showOpenDialog(JoinVideo.this);
```

```
        if (returnVal == JFileChooser.APPROVE_OPTION) {
                    direct=fc.getSelectedFile();
                    path.setText(""+direct.getPath());
                    currentDirectory=direct;

                                if (direct.isDirectory()){
                    arxeio=direct.listFiles();
                    int m=arxeio.length;

                                for(int i=0; i<arxeio.length; i++){
                            if(arxeio[i].isDirectory()){
                                m--;
                            }
                    }
                    files=new File[m];
                    int k=0;
                    for(int i=0; i<arxeio.length; i++){

                                if(arxeio[i].isDirectory()){
                                }
                                else{files[k]=arxeio[i];
                                k++;
                                }
                    }
               }else{
                    files=new File[1];
                                files[0]=direct;
                                }
                                combos(files);
               }

               }}
               );

    browse2.addActionListener(new ActionListener(){
               public void actionPerformed(ActionEvent e){
               if(currentDirectory==null){
                                fc2=new JFileChooser("c:/");
               }else {
                                fc2=new JFileChooser(currentDirectory);
               }
                 fc2.setFileHidingEnabled(true);
                                fc2.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
              fc2.setMultiSelectionEnabled(false);
              fc2.setDragEnabled(true);

                  int returnVal = fc2.showOpenDialog(JoinVideo.this);
            if (returnVal == JFileChooser.APPROVE_OPTION) {
                    direct1=fc2.getSelectedFile();
                    path1.setText(""+direct1.getPath());
                    currentDirectory=direct1;

                                if (direct1.isDirectory()){
                    arxeio1=direct1.listFiles();
                    int n=arxeio1.length;
                                for(int i=0; i<arxeio1.length; i++){
                            if(arxeio1[i].isDirectory()){
                                n--;
                            }
```

```
                }
                files1=new File[n];
                int k=0;
                for(int i=0; i<arxeio1.length; i++){

                        if(arxeio1[i].isDirectory()){
                        }
                        else{files1[k]=arxeio1[i];
                        k++;
                        }
                }
        }else{
            files1=new File[1];
                files1[0]=direct1;
                }
                combos1(files1);
        }

        }}
        );


browse1.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){

                        if(currentDirectory1==null){
                        fc1=new JFileChooser("c:/");
                }else {
                        fc1=new JFileChooser(currentDirectory1);
                                }
                                fc1.setFileHidingEnabled(true);
                   fc1.setFileSelectionMode(JFileChooser.FILES_ONLY);
          int returnVal = fc1.showSaveDialog(JoinVideo.this);
    if (returnVal == JFileChooser.APPROVE_OPTION) {
                sav=fc1.getSelectedFile();
                currentDirectory1=sav.getParentFile();
                s="file:"+sav.getPath();
        lista.output="file:"+sav.getPath();
                c.setText("Target: "+sav.getPath());
 }

        }}
        );

  }

void combos(File []arxeio){
        if(type=="icon"){
                  lista.comboMovies.removeAllItems();
                    path1.setText("Folder with the images                          ");
                    lista.listModel.removeAllElements();
                    lista.go.setEnabled(false);
                    lista.addButton.setEnabled(false);
                    lista.addallButton.setEnabled(false);
  }
  type="video";
  lista.type="video";
  this.arxeio=arxeio;
  for(int i=0;i<arxeio.length;i++){
```

```
 lista.comboMovies.addItem(arxeio[i].getPath());
 }

}

   void combos1(File []arxeio1){
       if(type=="video"){
                  lista.comboMovies.removeAllItems();
                    path.setText("Folder with the videos                                    ");
                    lista.listModel.removeAllElements();
                    lista.go.setEnabled(false);
                    lista.addButton.setEnabled(false);
                    lista.addallButton.setEnabled(false);
  }
  type="icon";
  lista.type="icon";
  this.arxeio1=arxeio1;
  for(int i=0;i<arxeio1.length;i++){
  lista.comboMovies.addItem(arxeio1[i].getPath());
 }
}


 private MenuBar createMenuBar() {
       ActionListener al = new ActionListener() {
          public void actionPerformed(ActionEvent ae) {
                String command = ae.getActionCommand();
                if (command.equals("Open Video")) {
                   if (fd == null) {
                         fd = new FileDialog(JoinVideo.this, "Open a video",
                                                    FileDialog.LOAD);
                         fd.setDirectory("/movies");
                   }
                   fd.show();
                   if (fd.getFile() != null) {
                         String filename = fd.getDirectory() + fd.getFile();
                         openFile("file:" + filename);
                   }
                } else if (command.equals("Exit")) {
                   dispose();
                   System.exit(0);
                }
                 else if (command.equals("New")) {
                   path.setText("Folder with the videos                                  ");
                   path1.setText("Folder with the images                                 ");
                   frameRate.setText("1");
                   width.setText("");
                   height.setText("");
                   lista.comboMovies.removeAllItems();
                   lista.listModel.removeAllElements();
                   lista.go.setEnabled(false);
                   lista.addButton.setEnabled(false);
                   lista.addallButton.setEnabled(false);
                }
                  else if (command.equals("Video codecs")){
          Codecs codecs = new Codecs();
                }
                         else if (command.equals("Play output file")){
    openFile(s);
```

```
                }
                                        else if (command.equals("Open Image")){
                    if (fd1 == null) {
                        fd1 = new FileDialog(JoinVideo.this, "Open an Image",
                                                        FileDialog.LOAD);
                                            }
                    fd1.show();
                    if (fd1.getFile() != null) {
                        String filename1 = fd1.getDirectory() + fd1.getFile();
                                OpenImage open= new OpenImage(filename1);
                    }
                }
                else if(command.equals("About")){
          JOptionPane.showMessageDialog
          (JoinVideo.this,"Made by Kampitakis Nektarios. All Rights Reserved.\nTEI
of Crete\nHeraclion December 2007\nMade with: Java - Java Media Framework",
                        "Java Multimedia Joiner",JOptionPane.PLAIN_MESSAGE);
          }
            }
        };

        MenuItem item;
        MenuBar mb = new MenuBar();

        Menu mnFile = new Menu("File");
        mnFile.add(item = new MenuItem("New"));
        item.addActionListener(al);
        mnFile.add(item = new MenuItem("Open Image"));
        item.addActionListener(al);
        mnFile.add(item = new MenuItem("Open Video"));
        item.addActionListener(al);
        mnFile.add(item = new MenuItem("Exit"));
        item.addActionListener(al);

        Menu mnOptions = new Menu("Options");
        mnOptions.add(item=new MenuItem("Video codecs"));
        item.addActionListener(al);
        mnOptions.add(item=new MenuItem("Play output file"));
        item.addActionListener(al);

        Menu mnAbout=new Menu("Help");
        mnAbout.add(item=new MenuItem("About"));
        item.addActionListener(al);

        mb.add(mnFile);
        mb.add(mnOptions);
        mb.add(mnAbout);
        return mb;
    }


  public void openFile(String filename) {
        String mediaFile = filename;
        Player player = null;

        URL url = null;
        try {

            if ((url = new URL(mediaFile)) == null) {
```

```
                Err("Can't build URL for " + mediaFile);
                return;
            }
            try {
                player = Manager.createPlayer(url);
            } catch (NoPlayerException e) {
                Err("Error: " + e);
            }
        } catch (MalformedURLException e) {
            Err("Error:" + e);
        } catch (IOException e) {
            Err("Error:" + e);
        }
        if (player != null) {
            this.filename = filename;
            JMFrame jmframe = new JMFrame(player, filename);
        }
    }
}
```

### *Κώδικας κλάσης JoinImages*

```
import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.Dimension;
import java.awt.image.*;
import java.awt.geom.AffineTransform;
import java.awt.geom.*;
import javax.media.*;
import javax.media.control.*;
import javax.media.protocol.*;
import javax.media.protocol.DataSource;
import javax.media.datasink.*;
import javax.media.format.VideoFormat;
import javax.media.format.*;
import javax.media.util.*;
import javax.imageio.*;


public class JoinImages implements ControllerListener, DataSinkListener {

    String []images;
    String out;
    String er,er1 ;

public JoinImages(int width,int height,float frameRate,String [] images,String out){

                Vector inputURL = new Vector();
        String outputURL = null;
this.images=images;
this.out=out;

        int i = 0;
        while (i < images.length) {
                inputURL.insertElementAt(images[i],i);
            i++;
            }
```

```
        outputURL=out;

                MediaLocator iml[] = new MediaLocator[images.length];
        MediaLocator oml;

        for (i = 0; i < images.length; i++) {
            if ((iml[i] = createMediaLocator(images[i])) == null) {
            }
        }

  oml=createMediaLocator(outputURL);
er="Failed to join the images!";

if(!doIt(width, height, frameRate, inputURL, oml)){

                JoinVideo.frr.dispose();
    JoinVideo.Err(er);
}
        }


public boolean doIt(int width, int height, float frameRate, Vector inFiles,
MediaLocator outML) {

                for(int i=0;i<inFiles.size();i++){

          String imageFile = (String)inFiles.elementAt(i);
                                BufferedImage img = null;
try {
    img = ImageIO.read(new File(imageFile));
} catch (IOException e) {
}
                try{
      if(img.getWidth()==0 || img.getHeight()==0){
                 er="Error: Wrong input image.";
                 return false;
                        }
                } catch (Exception e){
                        er="Error: "+e+" \nWrong input image";
                        return false;

                        }
}

        ImageDataSource ids = new ImageDataSource(width, height, frameRate,
inFiles);

        Processor p;

        try {
            p = Manager.createProcessor(ids);
        } catch (Exception e) {
            er="Cannot create a processor from the data source.";
            return false;
        }

        p.addControllerListener(this);
```

```
        p.configure();
        if (!waitForState(p, p.Configured)) {
           er="Failed to configure the processor.";
            return false;
        }

        ContentDescriptor cd;

        if ((cd = fileExtToCD(outML.getRemainder())) == null) {

                        er="Couldn't figure out from the file extension the type of
output needed!";
                        return false;
        }

        p.setContentDescriptor(cd);

        TrackControl tcs[] = p.getTrackControls();
        Format f[] = tcs[0].getSupportedFormats();
        if (f == null || f.length <= 0) {
            er="The mux does not support the input format: " + tcs[0].getFormat();
            return false;
        }

        tcs[0].setFormat(f[0]);

        p.realize();
        if (!waitForState(p, p.Realized)) {
            er="Failed to realize the processor.";
            return false;
        }

        //create a DataSink.
        DataSink dsink;
        if ((dsink = createDataSink(p, outML)) == null) {
            er="Failed to create a DataSink for the given output MediaLocator.";
            return false;
        }

        dsink.addDataSinkListener(this);
        fileDone = false;

        try {
            p.start();
            dsink.start();
        } catch (IOException e) {
            er="IO error during processing";
            return false;
        }

        // Wait for EndOfStream event.

        waitForFileDone();

        // Cleanup.
        try {
           dsink.close();
        } catch (Exception e) {}
        p.removeControllerListener(this);
```

```
JoinVideo.frr.setVisible(false);
JoinVideo.ok("Join has be done successfully!");

        return true;
    }


    /**
     * Create the DataSink.
     */
    DataSink createDataSink(Processor p, MediaLocator outML) {

        DataSource ds;

        if ((ds = p.getDataOutput()) == null) {
                    return null;
        }

        DataSink dsink;

        try {
            dsink = Manager.createDataSink(ds, outML);
            dsink.open();
        } catch (Exception e) {
            er=""+e;
            return null;
        }

        return dsink;
    }


    Object waitSync = new Object();
    boolean stateTransitionOK = true;


    boolean waitForState(Processor p, int state) {
        synchronized (waitSync) {
            try {
                while (p.getState() < state && stateTransitionOK)
                    waitSync.wait();
            } catch (Exception e) {}
        }
        return stateTransitionOK;
    }


    public void controllerUpdate(ControllerEvent evt) {

        if (evt instanceof ConfigureCompleteEvent ||
            evt instanceof RealizeCompleteEvent ||
            evt instanceof PrefetchCompleteEvent) {
            synchronized (waitSync) {
                stateTransitionOK = true;
                waitSync.notifyAll();
            }
        } else if (evt instanceof ResourceUnavailableEvent) {
            synchronized (waitSync) {
```

```
                stateTransitionOK = false;
                waitSync.notifyAll();
            }
        } else if (evt instanceof EndOfMediaEvent) {
            evt.getSourceController().stop();
            evt.getSourceController().close();
        }
    }


    Object waitFileSync = new Object();
    boolean fileDone = false;
    boolean fileSuccess = true;

    /**
     * Block until file writing is done.
     */
    boolean waitForFileDone() {
        synchronized (waitFileSync) {
            try {
                while (!fileDone)
                    waitFileSync.wait();
            } catch (Exception e) {}
        }
        return fileSuccess;
    }

    /**
     * Event handler for the file writer.
     */
    public void dataSinkUpdate(DataSinkEvent evt) {

        if (evt instanceof EndOfStreamEvent) {
            synchronized (waitFileSync) {
                fileDone = true;
                waitFileSync.notifyAll();
            }
        } else if (evt instanceof DataSinkErrorEvent) {
            synchronized (waitFileSync) {
                fileDone = true;
                fileSuccess = false;
                waitFileSync.notifyAll();
            }
        }
    }

ContentDescriptor fileExtToCD(String name) {

        String ext;
        int p;
        if ((p = name.lastIndexOf('.')) < 0)
            return null;
        ext = (name.substring(p + 1)).toLowerCase();
        String type;

 if (ext.equals("mov")){
     type=FileTypeDescriptor.QUICKTIME;
   }
   else if(ext.equals("avi")){
```

```
      type=FileTypeDescriptor.MSVIDEO;
     }
      else
      {
           if ((type = com.sun.media.MimeManager.getMimeType(ext)) == null)
                return null;
           type = ContentDescriptor.mimeTypeToPackageName(type);
        }

        return new FileTypeDescriptor(type);
   }

   /**
    * Create a media locator from the given string.
    */
   static MediaLocator createMediaLocator(String url) {
        MediaLocator ml;

        if (url.indexOf(":") > 0 && (ml = new MediaLocator(url)) != null)
           return ml;
        return null;
   }
}
```

## *Κώδικας κλάσης Join*

```
import java.awt.*;
import java.util.Vector;
import java.io.File;
import javax.media.*;
import javax.media.control.TrackControl;
import javax.media.control.QualityControl;
import javax.media.Format;
import javax.media.format.*;
import javax.media.datasink.*;
import javax.media.protocol.*;
import javax.media.protocol.DataSource;
import java.io.IOException;

public class Join implements ControllerListener, DataSinkListener {

   static int AUDIO = 0;
   static int VIDEO = AUDIO + 1;
   static int MEDIA_TYPES = VIDEO + 1;
   String []ur;
   String output;
   String er,er1 ;
   //int totalTracks;
   static String []aud;
   static String []vid;
   static String outAud,outVid;
   Processor ps;
   boolean transcodeMsg = false;


public Join(String []ur,String output){
        Vector inputURL = new Vector();
        String outputURL = null;
this.ur=ur;
```

```
this.output=output;


        outputURL=output;
        int i = 0;
        while (i < ur.length) {
                inputURL.insertElementAt(ur[i],i);
           i++;
           }

        MediaLocator iml[] = new MediaLocator[ur.length];
        MediaLocator oml;

        for (i = 0; i < ur.length; i++) {
           if ((iml[i] = createMediaLocator(ur[i])) == null) {
                JoinVideo.Err("Cannot build media locator from: " + ur[i]);
           }
        }
  oml=createMediaLocator(outputURL);

        er="Failed to join the videos.";
        if (!doIt(iml, oml)) {
                        JoinVideo.frr.dispose();
                JoinVideo.Err(er);
        }
}

 public boolean doIt(MediaLocator inML[], MediaLocator outML) {

        ContentDescriptor cd;

        if ((cd = fileExtToCD(outML.getRemainder())) == null) {

                        er="Couldn't figure out from the file extension the type of
output needed!";
           return false;
        }

        ProcInfo pInfo[] = new ProcInfo[inML.length];

        for (int i = 0; i < inML.length; i++) {
           pInfo[i] = new ProcInfo();
           pInfo[i].ml = inML[i];

           try {
                pInfo[i].p = Manager.createProcessor(inML[i]);
           } catch (Exception e) {
                er="Cannot create a processor from the given url: \n" + e;
                return false;
           }
        }


        if (!matchTracks(pInfo, cd)) {
           er="Failed to match the tracks.\n"+er1;
           return false;
        }

        if (!buildTracks(pInfo)) {
```

```
            er="Failed to build processors for the inputs\n"+er1;
            return false;
        }


        SuperGlueDataSource ds = new SuperGlueDataSource(pInfo);


        Processor p;
        try {
            p = Manager.createProcessor(ds);
        } catch (Exception e) {
            er="Failed to create a processor to join the inputs.";
            return false;
        }

        p.addControllerListener(this);


        if (!waitForState(p, p.Configured)) {
            er="Failed to configure the processor.";
            return false;
        }


        if ((p.setContentDescriptor(cd)) == null) {
            er="Failed to set the output content descriptor on the processor.";
            return false;
        }

        if (!waitForState(p, p.Realized)) {
            er="Failed to realize the processor.";
            return false;
        }


        DataSink dsink;
        if ((dsink = createDataSink(p, outML)) == null) {
            er="Failed to create a DataSink for the given output MediaLocator: " +
outML+"\n"+er1 ;
            return false;
        }

        dsink.addDataSinkListener(this);
        fileDone = false;



        try {
            p.start();
            dsink.start();
        } catch (IOException e) {
            er="IO error during concatenation";

            return false;
        }

        waitForFileDone();
```

```
        try {
            dsink.close();
        } catch (Exception e) {}
        p.removeControllerListener(this);

JoinVideo.frr.setVisible(false);
JoinVideo.ok("Join has be done successfully!");

//see output file format

            TrackControl track[];
            try {
                  ps = Manager.createProcessor(outML);
            } catch (Exception e) {
            }

        waitForState(ps,ps.Configured);
  track = ps.getTrackControls();
                     for (int j = 0; j < track.length; j++) {
                  if (track[j].getFormat() instanceof AudioFormat) {

        outAud=track[j].getFormat().toString();
                  } else if (track[j].getFormat() instanceof VideoFormat) {
                              outVid=track[j].getFormat().toString();
                  }
            }


        return true;

}

    public boolean matchTracks(ProcInfo pInfo[], ContentDescriptor cd) {

        TrackControl tcs[];

        Vector aTracks, vTracks;
        //int aIdx, vIdx;
        int i, j, type;
        TrackInfo tInfo;

                  aud=new String[pInfo.length];
          vid=new String[pInfo.length];
        for (i = 0; i < pInfo.length; i++) {

            if (!waitForState(pInfo[i].p, pInfo[i].p.Configured)) {
                  er1="- Failed to configure the processor.";
                  return false;
            }


            tcs = pInfo[i].p.getTrackControls();
            pInfo[i].tracksByType = new TrackInfo[MEDIA_TYPES];
            //pInfo[i].numTracksByType = new int[MEDIA_TYPES];
           // aIdx = vIdx = 0;


            for (j = 0; j < tcs.length; j++) {
```

```
                if (tcs[j].getFormat() instanceof AudioFormat) {

        aud[i]=tcs[j].getFormat().toString();
                    tInfo = new TrackInfo();
                    tInfo.idx = j;
                    tInfo.tc = tcs[j];
                    pInfo[i].tracksByType[AUDIO] = tInfo;
                  //  aIdx++;
                } else if (tcs[j].getFormat() instanceof VideoFormat) {
                        vid[i]=tcs[j].getFormat().toString();
                    tInfo = new TrackInfo();
                    tInfo.idx = j;
                    tInfo.tc = tcs[j];
                    pInfo[i].tracksByType[VIDEO] = tInfo;
                   // vIdx++;
                }
            }
             //pInfo[i].numTracksByType[AUDIO] = aIdx;
             //pInfo[i].numTracksByType[VIDEO] = vIdx;
            pInfo[i].p.setContentDescriptor(cd);
        }
                for (type = AUDIO; type < MEDIA_TYPES; type++) {
           // totalTracks += pInfo[0].numTracksByType[type];


        }
                for (type = AUDIO; type < MEDIA_TYPES; type++) {
                if (!tryMatch(pInfo, type)) {
                    er1="Cannot transcode the tracks to a common format for
concatenation!  Sorry.";
                    return false;
                }
        }
        return true;
    }

    public boolean buildTracks(ProcInfo pInfo[]) {

        ContentDescriptor cd = new ContentDescriptor(ContentDescriptor.RAW);
        Processor p;

        for (int i = 0; i < pInfo.length; i++) {
            p = pInfo[i].p;
            p.setContentDescriptor(cd);


            if (!waitForState(p, p.Realized)) {
                er1=" Failed to realize the processor.";
                return false;
            }


            PushBufferStream pbs[];
            TrackInfo tInfo;

            pInfo[i].ds = (PushBufferDataSource)p.getDataOutput();
            pbs = (PushBufferStream [])pInfo[i].ds.getStreams();
```

```
            for (int type = AUDIO; type < MEDIA_TYPES; type++) {

                    tInfo = pInfo[i].tracksByType[type];
                    tInfo.pbs = pbs[tInfo.idx];
            }
        }

        return true;
    }


    public boolean tryMatch(ProcInfo pInfo[], int type) {
        TrackControl tc;
  try{
           tc = pInfo[0].tracksByType[type].tc;
     }catch (Exception e){
          er1+=""+e;
          return false;
           }

        Format origFmt = tc.getFormat();
        Format newFmt, oldFmt;
        Format supported[] = tc.getSupportedFormats();

        for (int i = 0; i < supported.length; i++) {

            if (supported[i] instanceof AudioFormat) {
                // If it's not the original format, then for audio, we'll
                // only do linear since it's more accurate to compute the
                // audio times.
                if (!supported[i].matches(tc.getFormat()) &&

!supported[i].getEncoding().equalsIgnoreCase(AudioFormat.LINEAR))
                    continue;
            }

            if (tryTranscode(pInfo, 1, type, supported[i])) {

                // We've found the right format to transcode all the
                // tracks to.  We'll set it on the corresponding
                // TrackControl on each processor.

                for (int j = 0; j < pInfo.length; j++) {
                    tc = pInfo[j].tracksByType[type].tc;
                    oldFmt = tc.getFormat();
                    newFmt = supported[i];

                    // Check if it requires transcoding.
                    if (!oldFmt.matches(newFmt)) {
                        if (!transcodeMsg) {
                            transcodeMsg = true;
                        }
                    }

                    // For video, check if it requires scaling.
                    if (oldFmt instanceof VideoFormat) {
                        Dimension newSize = ((VideoFormat)origFmt).getSize();
                        Dimension oldSize = ((VideoFormat)oldFmt).getSize();
```

```
                        if (oldSize != null && !oldSize.equals(newSize)) {
                            // It requires scaling.

                            if (!transcodeMsg) {
                                transcodeMsg = true;
                            }
                            newFmt = (new VideoFormat(null,
                                        newSize,
                                        Format.NOT_SPECIFIED,
                                        null,
                                        Format.NOT_SPECIFIED)).intersects(newFmt);
                        }
                    }
                    tc.setFormat(newFmt);
                }

                return true;
            }
        }

        return false;

    }


    /**
     * Try different transcoded formats for concatenation.
     */
    public boolean tryTranscode(ProcInfo pInfo[], int procID, int type, Format
candidate) {
try{
        if (procID >= pInfo.length)
            return true;

        boolean matched = false;
        TrackControl tc = pInfo[procID].tracksByType[type].tc;
        Format supported[] = tc.getSupportedFormats();

        for (int i = 0; i < supported.length; i++) {
            if (candidate.matches(supported[i]) &&
                tryTranscode(pInfo, procID+1, type,candidate)) {
                matched = true;
                break;
            }
        }

        return matched;
} catch (Exception e){
        return false;
}
    }



    boolean isRawAudio(TrackInfo tInfo) {
        Format fmt = tInfo.tc.getFormat();
        return (fmt instanceof AudioFormat) &&
                fmt.getEncoding().equalsIgnoreCase(AudioFormat.LINEAR);
    }
```

```
public class StateWaiter implements ControllerListener {

    Processor p;
    boolean error = false;

    StateWaiter(Processor p) {
        this.p = p;
        p.addControllerListener(this);
    }

    public synchronized boolean waitForState(int state) {

        switch (state) {
        case Processor.Configured:
            p.configure(); break;
        case Processor.Realized:
            p.realize(); break;
        case Processor.Prefetched:
            p.prefetch(); break;
        case Processor.Started:
            p.start(); break;
        }

        while (p.getState() < state && !error) {
            try {
                wait(1000);
            } catch (Exception e) {
            }
        }
        return !(error);
    }

    public void controllerUpdate(ControllerEvent ce) {
        if (ce instanceof ControllerErrorEvent) {
            error = true;
        }
        synchronized (this) {
            notifyAll();
        }
    }
}


DataSink createDataSink(Processor p, MediaLocator outML) {

    DataSource ds;

    if ((ds = p.getDataOutput()) == null) {
        er1=" The processor does not have an output DataSource";
        return null;
    }

    DataSink dsink;

    try {
        dsink = Manager.createDataSink(ds, outML);
        dsink.open();
```

```
        } catch (Exception e) {
            er1="Cannot create the DataSink: " + e;
            return null;
        }

        return dsink;
}


boolean waitForState(Processor p, int state) {
        return (new StateWaiter(p)).waitForState(state);
}


public void controllerUpdate(ControllerEvent evt) {

        if (evt instanceof ControllerErrorEvent) {
           JoinVideo.Err("Failed to join the files.");
        } else if (evt instanceof EndOfMediaEvent) {
            evt.getSourceController().close();
        }
}

Object waitFileSync = new Object();
boolean fileDone = false;
boolean fileSuccess = true;


boolean waitForFileDone() {
        synchronized (waitFileSync) {

            try {
                    while (!fileDone) {
                        waitFileSync.wait(1000);


                    }
            } catch (Exception e) {}
        }

        return fileSuccess;
}



public void dataSinkUpdate(DataSinkEvent evt) {

        if (evt instanceof EndOfStreamEvent) {
            synchronized (waitFileSync) {
                    fileDone = true;
                    waitFileSync.notifyAll();
            }
        } else if (evt instanceof DataSinkErrorEvent) {
            synchronized (waitFileSync) {
                    fileDone = true;
                    fileSuccess = false;
                    waitFileSync.notifyAll();
            }
        }
```

```
    }

    ContentDescriptor fileExtToCD(String name) {

        String ext;
        int p;


        if ((p = name.lastIndexOf('.')) < 0)
            return null;

        ext = (name.substring(p + 1)).toLowerCase();

        String type;
             if (ext.equals("mov")){
      type=FileTypeDescriptor.QUICKTIME;
     }
       else
            if (ext.equals("avi")){
      type=FileTypeDescriptor.MSVIDEO;
       }
       else

             if ((type = com.sun.media.MimeManager.getMimeType(ext)) == null){
                   return null;
          }
             type = ContentDescriptor.mimeTypeToPackageName(type);


        return new FileTypeDescriptor(type);
    }


    static MediaLocator createMediaLocator(String url) {

        MediaLocator ml;

        if (url.indexOf(":") > 0 && (ml = new MediaLocator(url)) != null)
            return ml;

        return null;
    }
}
```