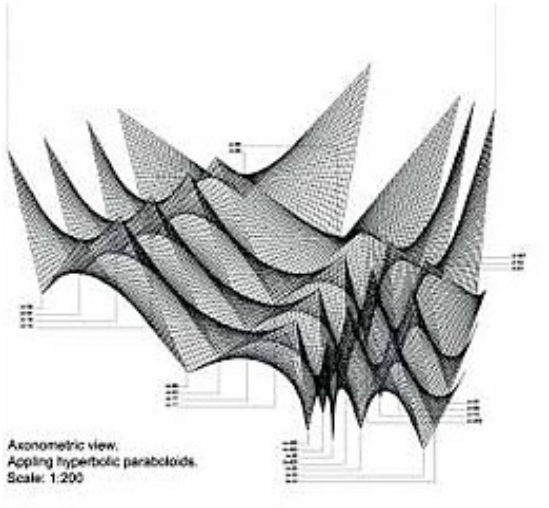
Τ.Ε.Ι. ΚΡΗΤΗΣ

ΤΜΗΜΑ ΜΟΥΣΙΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ ΚΑΙ ΑΚΟΥΣΤΙΚΗΣ

Πτυχιακή Εργασία

"Ανάπτυξη εφαρμογών σύνθεσης ήχου στο περιβάλλον προγραμματισμού (Application Programming Interface) JSyn της Java"

Γκακίδης Καριοφύλλης



Επίβλεψη: Ταξιάρχης Διαμαντόπουλος

Ρέθυμνο, Ιούνιος 2008

Περιεχόμενα

Εισαγωγή

Το JSyn είναι ένα Application Programming Interface (API) το οποίο επιτρέπει την ανάπτυξη διαδραστικών μουσικών προγραμμάτων με την χρήση της γλώσσας προγραμματισμού Java. Μπορεί κάποιος να τρέξει αυτά τα προγράμματα ως ανεξάρτητες εφαρμογές ή σαν applets τα οποία τρέχουν μέσα σε κάποιο web browser. Για να τρέξουν αυτά τα προγράμματα σε οποιοδήποτε σύστημα πρέπει να έχει εγκατασταθεί και το JSyn plug-in σε αυτό. Οι εφαρμογές που μπορεί να γίνουν με το JSyn αφορούν σε ηχητικά εφφέ, ηχητικά περιβάλλοντα και μουσική. Είναι βασισμένο στο παραδοσιακό μοντέλο ηχητικών μονάδων-γεννητριών οι οποίες συνδεδεμένες μεταξύ τους μπορούν να παράγουν πιο περίπλοκους ήχους.

Το JSyn το ανέπτυξε ο Phill Burk, και το παρουσίασε πρώτη φορά στο ICMC (International Computer Music Conference) το 1998. Από εκεί και μετά το JSyn συνεχίζει να αναπτύσσεται με εκδόσεις σχεδόν κάθε χρόνο. Η τελευταία έκδοση έγινε τον Ιανουάριο του 2008.

Προς το παρόν το JSyn plug-in υποστηρίζει δύο λειτουργικά συστήματα. Το Microsoft Windows μέχρι την τελευταία έκδοση Vista, και το Apple Mac OS X μέχρι την τελευταία έκδοση Leopard 10.5. Η υποστήριξη για το λειτουργικό Linux θα είναι εφικτή στο άμεσο μέλλον.

Στην εργασία αυτή παρουσιάζονται οι λειτουργίες του JSyn και πώς αυτές βοηθούν στην ανάπτυξη εφαρμογών σύνθεσης ήχου. Ακόμα κάποιος μπορεί να δει τις εσωτερικές διεργασίες που γίνονται ώστε να μετατραπεί ο κώδικας γραμμένος σε Java μέχρι να φτάσει στην δυαδική μορφή του. Παρουσιάζονται ακόμα οι βασικές κλάσεις και οι κύριες ηχητικές μονάδες που παρέχει το JSyn.

Επίσης έχουν αναπτυχθεί τρεις εφαρμογές με χρήση του JSyn και της Java. Με αυτές γίνεται προσπάθεια να αναπτυχθεί ένα ευρύ φάσμα τεχνικών σύνθεσης ήχου ώστε να γίνει αντιληπτό το μέγεθος και η ποιότητα του πακέτου JSyn. Στην πρώτη εφαρμογή χρησιμοποιούνται οι τεχνικές σύνθεσης βασισμένες στην δειγματοληψία, στις σειρές καθυστέρησης και στην αφαιρετική σύνθεση. Στην δεύτερη εφαρμογή χρησιμοποιείται η σύνθεση πίνακα κυματομορφής, η προσθετική σύνθεση, και η τεχνική διαμόρφωσης πλάτους. Στην τρίτη γίνεται χρήση της τεχνικής σύνθεσης ήχου διαμόρφωσής συχνότητας.

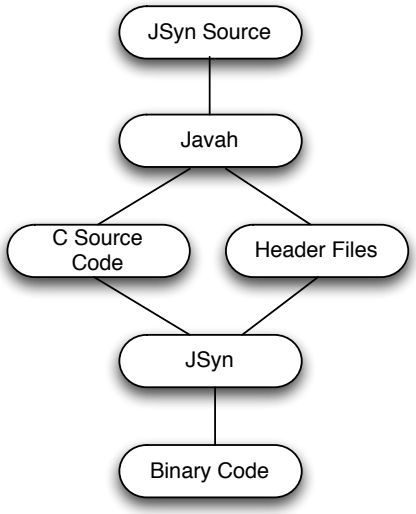
Όλες οι εφαρμογές προσφέρουν στον χρήστη έλεγχο σε μερικές βασικές παραμέτρους με την χρήση γραφικού περιβάλλοντος διάδρασης σχεδιασμένο στην γλώσσα προγραμματισμού Java.

Κεφάλαιο 1: JSyn

Το JSyn προσφέρει, μέσω της γλώσσας προγραμματισμού Java, σύνθεση ήχου πραγματικού χρόνου, βασισμένη σε ηχητικές μονάδες-γεννήτριες για "stand-alone"[1] εφαρμογές ή διαδυκτιακά applets προγραμματισμένα στην γλώσσα Java. Οι ηχητικές μονάδες του JSyn μπορούν να συνδεθούν άμεσα μεταξύ τους, ώστε να επιτρέπεται ο δυναμικός χειρισμός της γλώσσας για την δημιουργία πολύπλοκων μηχανών σύνθεσης ήχου.

1.1 Πίσω από το JSyn

Η μηχανή του JSyn για την αναπαραγωγή ήχου έχει προγραμματιστεί με την χρήση native κώδικα στην γλώσσα C. Ο κώδικας αυτός συναντιέται είτε σαν ένα Netscape plug-in (applets), είτε σαν ένα dll αρχείο(windows), είτε σαν ένα αρχείο τύπου extension(MacOS), είτε σαν σε μια shared library(Unix) αναλόγως με την πλατφόρμα στην οποία εκτελείται η εφαρμογή. Ο κώδικας της JSyn μαζί με τα libraries μεταφράζονται κάθε φορά από έναν ειδικό μεταφραστή, τον javah[2], ο οποίος αναλαμβάνει να παράγει τον αντίστοιχο native πηγαίο κώδικα σε C καθώς και τα απαιτούμενα header files.



Ο λόγος για τον οποίο γίνεται αυτή η μετατροπή είναι η σταθερότητα, η ακρίβεια και η ταχύτητα της γλώσσας C η οποία παρέχει την απαιτούμενη ακρίβεια σε audio εφαρμογές. Όπως είναι κατανοητό σε εφαρμογές ψηφιακής σύνθεσης ήχου με την δειγματοληψία να φτάνει στα 44100 και στα 96000 δείγματα ανά δευτερόλεπτο είναι αναγκαία η ακρίβεια και η σταθερότητα για την ακέραιη αναπαραγωγή του

[1] Εφαρμογές "stand-alone": Οι εφαρμογές που τρέχουν σε ένα σύστημα και δεν χρειάζεται να εγκατασταθούν σ'αυτό.

[2] javah: Εφαρμογή που περιέχεται στο Java Development Kit. Δημιουργεί header files και κώδικα της γλώσσας C. Συμβάλει στην διασύνδεση και αλληλεπίδραση C και Java κώδικα.

ήχου. Άλλωστε στις αρχικές εκδόσεις τις Java υποστηριζόταν μόνο το βάθος ανάλυσης 8-bit και η δειγματοληψία 22 KHz.

1.2 Οι βασικές λειτουργίες του JSyn API

1.2.1 Δημιουργία Ηχητικών Μονάδων

Οι ηχητικές μονάδες που χρειάζεται ο προγραμματιστής για να δημιουργήσει την δικιά του μηχανή δημιουργούνται με την δημιουργία ενός αντικειμένου μίας από τις Java κλάσεις που είναι ορισμένες από το JSyn. Για παράδειγμα για να δημιουργήσει κάποιος μια γεννήτρια ημιτονοειδούς σήματος να το χρησιμοποιήσει στον περεταίρω σχεδιασμό του, αρκεί να χρησιμοποιήσει τον κώδικα που ορίζει τον constructor:

```
SineOscillator myOsc = new SineOscillator();
```

1.2.2 Αρχή και τέλος της λειτουργίας των μονάδων

Οι ηχητικές μονάδες για να λειτουργήσουν μέσα στο κύκλωμα που σχεδιάζει ο προγραμματιστής πρέπει να μπουν στην λίστα των μονάδων που θα εκτελεστούν από την μηχανή σύνθεσης ήχου. Αυτό γίνεται με την κλήση της μεθόδου start() την οποία έχουν κληρονομήσει όλες οι μονάδες από την μητρική κλάση του JSyn SynthUnit. Το ίδιο γίνεται και για τον τερματισμό της λειτουργίας μιας μονάδας, όταν πια δεν χρειάζεται στο κύκλωμα και καταναλώνει μνήμη και επεξεργαστική ισχύ που ίσως χρειάζεται το κύκλωμα για άλλες λειτουργίες. Η μέθοδος αυτή ονομάζεται stop() και αφαιρεί την μονάδα από τον κατάλογο των μονάδων που εκτελούνται από την μηχανή σύνθεσης ήχου. Η λειτουργία της γεννήτριας που δημιουργήθηκε παραπάνω μπορεί να αρχίσει και να τελειώσει με την παράθεση του παρακάτω κώδικα:

```
myOsc.start();
myOsc.stop();
```

1.2.3 Διασύνδεση μονάδων

Οι μονάδες ήχου παρέχουν επίσης εισόδους και εξόδους επικοινωνίας, είτε για το σήμα είτε για την προσαρμογή των παραμέτρων τους σε πραγματικό χρόνο. Αυτές οι θύρες επικοινωνίας μπορούν να συνδέονται μεταξύ τους για την δημιουργία κυκλωμάτων που περιέχουν πολλές μονάδες ήχου. Για να συνδέσει κάποιος για παράδειγμα την έξοδο ενός ταλαντωτή(μέθοδος output) με την είσοδο ενός φίλτρου(μέθοδος input) θα πρέπει να χρησιμοποιήσει την μέθοδο connect.

```
myOsc.output.connect( myFilter.input );
```

Οι έξοδοι των μονάδων μπορούν να συνδέονται με πολλαπλές εισόδους άλλων μονάδων. Αντιθέτως οι είσοδοι μπορούν να πάρουν σήμα μόνο από μια μοναδική έξοδο.
Εσωτερικά όλα τα σήματα που διαπερνάνε τις μονάδες ήχου είναι είτε RAW_SIGNED το οποίο είναι τύπος σήματος που μπορεί να παίρνει τιμές από -1.0 με 1.0, είτε RAW_UNSIGNED(χωρίς header) του οποίου το εύρος κυμαίνεται στις τιμές από 0.0

μέχρι 2.0 . Αυτό συμβαίνει για να μπορούν να υποστηριχθούν DSP επιταχυντές επεξεργασίας που δεν χρησιμοποιούν αρνητικές τιμές.

1.2.4 Ρύθμιση των παραμέτρων και του είδους σήματος των μονάδων.

Οι τιμές των παραμέτρων των μονάδων μπορούν να προκαθοριστούν από τον προγραμματιστή. Για παράδειγμα:

myOsc.frequency.set(fundamental * (3.0/2.0));

Η μέθοδος set() δέχεται τιμές σε μονάδες που είναι κατάλληλες για κάθε μία είσοδο της ηχητικής μονάδας. Για παράδειγμα η συχνότητα ενός ημιτονοειδούς ταλαντωτή δέχεται τιμές σε Hz. Άλλες ηχητικές μονάδες αναλόγως το είδος τους, μπορούν να δέχονται τύπους σήματος με εύρος τιμών για παραμέτρους όπως sample rate, filter cutoff, exponential lag decay rates, delay times κ.α. Ο τύπος σήματος που έχει προκαθοριστεί για κάθε θύρα επικοινωνίας της ηχητικής μονάδας καθορίζει το εύρος των επιτρεπόμενων τιμών και τον τρόπο που οι τιμές αυτές θα μετατραπούν ώστε να μπορούν να γίνουν επεξεργάσιμες από την μηχανή σύνθεσης ήχου.

Το JSyn παρέχει και ηχητικές μονάδες για τις αριθμητικές πράξεις μεταξύ των σημάτων. Κάποιος μπορεί να προσθέσει, να αφαιρέσει, να πολλαπλασιάσει ή να διαιρέσει δυο σήματα μεταξύ τους, πράγμα πολύ χρήσιμο για την παραγωγή τεχνικών σύνθεσης ήχου. Επίσης κάποιος μπορεί να κάνει οποιαδήποτε πράξη σε ένα σήμα παραθέτοντας έναν σταθερό αριθμό για την πράξη αυτή. Αρκεί να δηλωθεί ο τύπος του σήματος που έχει κατάλληλο εύρος για το μέγεθος του σταθερού αριθμού.

```
myAdder.inputB.setSignalType(Synth.SIGNAL_TYPE_OSC_FREQ );
myAdder.inputB.set( 300.0 );
```

1.2.5 Δίαυλοι σήματος (signal busses)

Το JSyn παρέχει διαύλους μίξης σήματος. Όταν υπάρχει ένας ικανός αριθμός σημάτων τα οποία θα πρέπει να μιξαριστούν σε κάποια φάση του κυκλώματος, με την χρήση ενός δίαυλου είναι δυνατή η αποφυγή χρήσης πολλών αντικειμένων τις κλάσης AddUnit, τα οποία θα επιβάρυναν την χρήση μνήμης και επεξεργαστικής ισχύος κάτι που γενικά πρέπει να αποφεύγεται από τους προγραμματιστές εφαρμογών.

Οι ηχητικές μονάδες που δέχονται τα σήματα προς μίξη είναι τα αντικείμενα της κλάσης BusWriter. Τα αντικείμενα της κλάσης BusReader μετατρέπουν τα κανονικά σήματα που δέχονται οι BusWriters σε σήματα διαύλου. Στην ουσία πραγματοποιούν την μίξη.

Οι BusReaders έχουν μία είσοδο, την SynthBusInput η οποία μπορεί να έχει πολλαπλές εξόδους SynthBusOutput συνδεδεμένες με αυτήν. Όλες οι έξοδοι SynthBusOutputs οι οποίες συνδέονται σε μία είσοδο SynthBusInput προσθέτονται και αυτές μεταξύ τους. Με τον τρόπο αυτό μπορεί να γίνει μίξη πολλαπλών διαύλων.

1.2.6 Τύποι δεδομένων – Δείγματα, Πίνακες, Envelopes

Το JSyn υποστηρίζει τρεις κλάσεις για την αποθήκευση δεδομένων(data containers) και είναι οι εξής: SynthSample, SynthTable, SynthEnvelope. Έχουν αναπτυχθεί η καθεμία για διαφορετικές χρήσεις, ανάλογα την επεξεργασία που χρειάζεται, καθώς στις τεχνικές σύνθεσης υπάρχουν διαφορετικές ανάγκες για πόρους του συστήματος. Μια εφαρμογή μπορεί να έχει πρόσβαση στα δεδομένα που βρίσκονται μέσα στις κλάσεις αυτές, μέσω των μεθόδων read() και write(). Τα δεδομένα δεν μπορούν να χρησιμοποιούνται από κάτι άλλο όταν τα χρησιμοποιεί η μηχανή σύνθεσης ήχου. Αυτό δίνει την δυνατότητα στο JSyn να αποθηκεύει τα ηχητικά δεδομένα σε ειδικές μνήμες, όπως η RAM σε μια κάρτα ήχου, ή σε μια τοπική στατική μνήμη από DSP.

Αντικείμενα SynthSample

Περιέχουν ακέραιους δεκαεξαδικούς αριθμούς που περιγράφουν αρχεία ήχου συνήθως μονοφωνικά ή στερεοφωνικά με βάθος ανάλυσης 16bit . Τα αντικείμενα της κλάσης SynthSample χρησιμοποιούνται για την αποθήκευση ψηφιακών ηχητικών δειγμάτων που έχουν διαβαστεί από ένα αρχείο .wav ή .aiff. Η πληροφορία στις θέσεις μνήμης μπορεί να γραφεί ή να διαβαστεί μόνο συνεχόμενα. Δεν μπορεί ο δείκτης διαβάσματος να πηγαίνει σε μη συνεχόμενες τιμές του index. Το διάβασμα ή το γράψιμο του buffer το αναλαμβάνουν οι SampleReader_16V1 και SampleWriter_16F1 αντίστοιχα. Αυτοί μπορούν να συνδυαστούν έτσι ώστε να μας δώσουν τεχνικές σύνθεσης με την χρήση σειρών καθυστέρησης. Όταν τα δείγματα ήχου που πρόκειται να διαβαστούν είναι μεγάλα σε χρόνο, δαπανούν μεγάλο μέρος της μνήμης του υπολογιστή ή της DRAM μνήμης μιας κάρτας ήχου.

Αντικείμενα SynthTable

Ένα αντικείμενο της κλάσης SynthTable μπορεί να περιέχει τα δείγματα του ήχου σε μορφή ακέραιου αριθμού ή αριθμού κινητής υποδιαστολής αν το σύστημα που φιλοξενεί την εφαρμογή του JSyn το υποστηρίζει. Οι πίνακες SynthTable μπορούν να διαβαστούν με τυχαία προσπέλαση του δείκτη διαβάσματος από μονάδες ήχου όπως οι WaveShaper, TableOscillator και μονάδες καθυστέρησης μικρού χρόνου. Οι πίνακες αυτοί είναι συνήθως μικροί και αποθηκεύονται συνήθως στην στατική RAM ενός υπολογιστή ή στο DSP μιας κάρτας ήχου.

Αντικείμενα SynthEnvelope

Ένας SynthEnvelope περιέχει ζευγάρια τιμών [αξίας-διάρκειας]. Μπορεί κανείς να διαβάσει αυτά τα ζευγάρια τιμών μόνο κατά αύξουσα σειρά θέσης μνήμης. Όπως και τα SynthSamples αποθηκεύονται στην κύρια μνήμη του υπολογιστή είτε στην στατική μνήμη ενός DSP. Οι SynthEnvelopes μπορούν να χρησιμοποιηθούν ως περιβάλλουσες παραμέτρων ή σαν breakpoint ενός control ταλαντωτή.

1.2.7 Σειρά εκτέλεσης

Οι εφαρμογές του JSyn μπορούν να βάλουν σε μια σειρά εκτέλεσης (queuing) ομάδες δεδομένων που παράγονται από ένα αντικείμενο SynthSample σε ένα άλλο αντικείμενο που ονομάζεται SynthSampleQueue που αποτελεί μια θύρα επικοινωνίας σε μια ηχητική μονάδα. Τα δεδομένα (συνήθως ηχητικά δείγματα) θα παίζονται με τη σειρά που ο χρήστης επιλέξει μέχρι να αδειάσει η λίστα της σειράς εκτέλεσης(queue). Υπάρχει και η επιλογή της επαναλαμβανόμενης εκτέλεσης του τελευταίου δείγματος ή ηχητικού συμβάντος που έχει οριστεί τελευταίο στην λίστα εκτέλεσης(queueLoop). Αν επιλεχθεί να μπει στη σειρά εκτέλεσης ένα νέο συμβάν

όσο επαναλαμβάνεται το τελευταίο, μόλις τελειώσει η κανονική του διάρκεια θα αρχίσει να επαναλαμβάνεται το νέο συμβάν. Παρακάτω υπάρχει ένα παράδειγμα κώδικα που περιγράφει την χρήση του queuing σε μια περιβάλλουσα έντασης.

```
mySampler.samplePort.queueLoop(mySamp);
```

1.2.8 Κυκλώματα ιεραρχικής δομής (patches)

Χρησιμοποιώντας ένας προγραμματιστής την κλάση SynthCircuit μέσα στην βασική του κλάση (sub-classing) μπορεί να φτιάξει στην δικιά του εφαρμογή πιο μικρά και απλά κυκλώματα σύνθεσης ήχου βάζοντας μέσα γεννήτριες και μονάδες ήχου, και συνθέτοντας τα μαζί σε ένα patch να φτιάξει με μεγαλύτερη ευελιξία μια πιο περίπλοκη μηχανή σύνθεσης. Ο προγραμματιστής μπορεί να ορίσει στα patches αυτά εισόδους-εξόδους και σημεία ελέγχου των παραμέτρων τους. Όταν το patch αυτό δεχθεί την εντολή να εκτελεστεί από την native μηχανή σύνθεσης ήχου τότε όλες οι μονάδες που περιέχει θα ξεκινήσουν να εκτελούνται και αυτές ταυτόχρονα μαζί με το κεντρικό patch,χωρίς την χρήση περεταίρω κώδικα για την εκκίνηση των μονάδων ήχου που περιέχει το κεντρικό patch

1.2.9 Χρονική Διαχείριση Γεγονότων

Η Java Virtual Machine δεν παρέχει στον προγραμματιστή έλεγχο με ακρίβεια σε πραγματικό χρόνο των γεγονότων που συμβαίνουν κατά την διάρκεια της λειτουργίας της εφαρμογής, κάτι που είναι απαραιτητο στην εκτέλεση μουσικών έργων. Το JSyn παρ' όλα αυτά παρέχει μια προσωρινή θέση μνήμης η οποία είναι διαθέσιμη για τον ακριβή έλεγχο των γεγονότων που συμβαίνουν μέσα στον χρόνο. Ο προγραμματιστής ή ο χρήστης μπορεί να ελέγξει έτσι το χρόνο εκκίνησης ή τερματισμού των ηχητικών μονάδων που χρησιμοποιεί, τον χρόνο αλλαγής παραμέτρων, τον χρόνο εκκίνησης της αναπαραγωγής δειγμάτων ήχου και γενικά κρίσιμες λειτουργίες μιας εφαρμογής μουσικού τύπου. Η μονάδα μέτρησης του JSyn είναι τα tick όπως ονομάστηκε από τους δημιουργούς του. Τα ticks αναλαμβάνει να ορίσει ένας ειδικός μετρητής ο οποίος αυξάνει την τιμή τους για κάθε 64 δείγματα που αναπαράγει η εφαρμογή JSyn. Ένα παράδειγμα κώδικα που γίνεται χρήση του μετρητή αυτού είναι το παρακάτω:

```
int time = Synth.getTickCount();
/* Start 300 ticks in the future. */
   myOsc.start( time + 300 );
/* Set frequency to 220 Hz 400 ticks after starting. */
   myOsc.frequency.setAt( time + 700, 220.0 );
```

1.3. Οι ηχητικές μονάδες που παρέχονται από JSyn

Το JSyn παρέχει στον προγραμματιστή ένα εύρος από έτοιμα κυκλώματα σύνθεσης ήχου ώστε να απλοποιήσει τον κώδικα των εφαρμογών του. Αυτά τα κυκλώματα ή οι μονάδες που εκτελούν άλλοτε απλές και άλλοτε πολυσύνθετες εργασίες, είναι Java κλάσεις και μπορούν να χρησιμοποιηθούν απλά καλώντας τα αντικείμενα τους.

Κάποιες κλάσεις για βασικά κυκλώματα σύνθεσης ήχου είναι παρακάτω:

Αριθμητικές και Λογικές Μονάδες

AddUnit - output = inputA + inputB, εύρος τιμών -1.0 ,1.0.
CompareUnit - output = (inputA > inputB) εύρος 1.0 : 0.0;
LatchUnit - output = (Gate > 0.0) , Input : previous-output; χρησιμοποιείται στην τεχνική sample and hold.
MaximumUnit - output = (inputA > inputB) ? inputA : inputB.
MinimumUnit - output = (inputA < inputB) ? inputA : inputB.
MultiplyUnit - output = inputA * inputB
MultiplyAddUnit - output = inputA * inputB + inputC
MultiplyAddUnsignedUnit - output = inputA * inputB + inputC. Η είσοδος C και η έξοδος είναι μη προσημασμένες.
MultiplyUnsignedUnit - output = inputA * inputB. Η είσοδος A,B και η έξοδος είναι μη προσημασμένες.
SchmidtTrigger – Συγκριτής με υστέρηση.(διαχωρισμένα Set και Reset επίπεδα).
SelectUnit - output = (select > 0.0) , inputA : inputB;
SubtractUnit - output = inputA- inputB, εύρος -1.0 ,1.0.

Μονάδες Ρυθμίσεων

CrossFade – Τεχνική Cross fade μεταξύ των δυο εισόδων.
ExponentialLag – Η έξοδος παίρνει την τιμή της εισόδου με εκθετική αύξηση ή μείωση.
EnvelopePlayer – Αναπαράγει γραμμικά τα σημεία μίας περιβάλλουσας.
FourWayCrossFade - Τεχνική Cross fade μεταξύ των τεσσάρων εισόδων.
PanUnit – Τοποθετεί ένα μονοφωνικό σήμα σε μια στερεφωνική μίξη δυο εξόδων.
ParabolicEnvelope –Παράγει μια τοξωτή περιβάλλουσα χρήσιμη στην granular σύνθεση.

Μονάδες Φίλτρων

Filter_1o1z - output = y(n) = A0*x(n) + A1*x(n-1), πρώτης τάξης.
Filter_1o1p - output = y(n) = A0*x(n) - B1*y(n-1), πρώτης τάξης, ενός πόλου.
Filter_1o1p1z - output = y(n) = A0*x(n) + A1*x(n-1) - B1*y(n-1), allpass φίλτρο.
Filter_2o2p - output = y(n) = A0*x(n) - B1*y(n-1) - B2*y(n-2), δεύτρης τάξης, δυο πόλων (reson)
Filter_2o2p2z - output = y(n) = 2.0 * (A0*x(n) + A1*x(n-1) + A2*x(n-2) -

B1*y(n-1) - B2*y(n-2)), biquad φίλτρο.

StateVariableFilter – Μεταβλητής τάξης Resonant φίλτρο με LowPass, BandPass και HighPass εξόδους.

Διάφορες Μονάδες

DelayUnit - output = Input καθυστερημένη για ορισμένο χρόνο

InterpolatingDelayUnit - output = Input καθυστερημένη για μεταβλητούς χρόνους.

BusReader – Έχει είσοδο τύπου Bus και κανονική έξοδο. Χρήσιμη για μίξη πολλών σημάτων.

BusWriter - Έχει κανονική είσοδο και έξοδο τύπου Bus. Χρήσιμη για μίξη πολλών σημάτων.

LineOut – Στέλνει το σήμα σε έναν γενικό μίκτη και μετά στο DAC.

WaveShaper – Συνδέει με interpolation τις τιμές ενός πίνακα.

Μονάδες Θορύβου

RedNoise – Παρεμβάλλεται γραμμικά μεταξύ των τιμών ενός θορύβου με έλεγχο συχνότητας.

WhiteNoise - output = Τυχαίες τιμές δειγμάτων που διανέμονται μεταξύ των τιμών -1.0 ,1.0.

Μονάδες Ταλαντωτών

ImpulseOscillator – Σειρά παλμών με έλεγχο συχνότητας και έντασης.

PulseOscillator – Ταλαντωτής παλαμοειδούς κυματομορφής με έλεγχο του εύρους του παλμού

SawtoothOscillator – Πριονωτή κυματομορφή.

SineOscillator – Ημιτονοειδής κυματομορφή.

SquareOscillator – Τετραγωνική κυματομορφή

TableOscillator – Αυθαίρετη κυματομορφή που διαβάζεται από έναν πίνακα. Ελέγχεται από την παράμετρο “συχνότητα” και όχι από τη συχνότητα δειγματοληψίας.

TriangleOscillator – Τριγωνοειδής κυματομορφή.

Μονάδες δειγματοληψίας

SampleReader_16F1 – Αναπαραγωγή δείγματος 16 bit mono

SampleReader_16F2 - Αναπαραγωγή δείγματος 16 bit stereo

SampleReader_16V1 - Αναπαραγωγή δείγματος 16 bit mono με μεταβαλλόμενη συχνότητα δειγματοληψίας και γραμμικό interpolation.

SampleWriter_16F1 - Η είσοδος εγγράφεται σε ένα αντικείμενο SynthSample και εξυπηρετεί τεχνικές σειρών καθυστέρησης.

1.4 Οι βασικές κλάσεις του JSyn

Το πακέτο JSyn αποτελείται, όπως όλα τα πακέτα της Java, από κλάσεις και interfaces. Οι δύο σημαντικότερες κλάσεις στις οποίες βασίζονται οι κυριότερες λειτουργίες του JSyn είναι η κλάση SynthObject και η SynthPort. Οι κλάσεις αυτές

είναι μητρικές κλάσεις (root classes) καθώς κληρονομούνε όλες τις μεθόδους και τα πεδία τους σε όλες τις επόμενες κλάσεις, που φτιάχτηκαν δηλαδή από αυτές.

Βασική κλάση είναι και η SynthContext, η οποία παρέχει στην εφαρμογή ένα πλαίσιο με σταθερή συχνότητα δειγματοληψίας για να πραγματοποιηθούν πάνω της τεχνικές σύνθεσης ήχου.

Μπορούν να χρησιμοποιηθούν πολλές τέτοιες βάσεις σε κάθε εφαρμογή έτσι ώστε κάθε τεχνική ήχου να χρησιμοποιεί την δικιά της συχνότητα δειγματοληψίας. Η κλάση αυτή δεν έχει κάποιες παράγωγές της. Τέλος η κλάση Synth χωρίς να έχει παράγωγες κλάσεις και χωρίς να προέρχεται από τις παραπάνω κλάσεις, θεωρείται βασική καθώς παρέχει συνεχή έλεγχο σε μια εφαρμογή και αντίθετα με την SynthContext μοιράζει ένα πλαίσιο που χρησιμοποιεί συγκεκριμένη συχνότητα δειγματοληψίας σε κάθε τεχνική που υπάρχει στην εφαρμογή. Οι υπόλοιπες κλάσεις που δεν προέρχονται από τις δύο παραπάνω κλάσεις είναι φτιαγμένες είτε για την δημιουργία μουσικών γραφικών περιβαλλόντων για τον έλεγχο των εφαρμογών (knobs, faders κ.α.) και είναι βασισμένες στο πακέτο της Java java.awt, είτε είναι γενικών καθηκόντων όπως για παράδειγμα η κλάση FileSearch που χρησιμοποιείται για εύρεση αρχείων, και η κλάση VoiceAllocator με την οποία ανιχνεύεται ο αριθμός των φωνών που χρησιμοποιείται κάθε φορά από μία εφαρμογή, ή χρησιμοποιούνται για πολύπλοκες μαθηματικές πράξεις όπως η ChebychevPolynomial ή η TablePolyonimal.

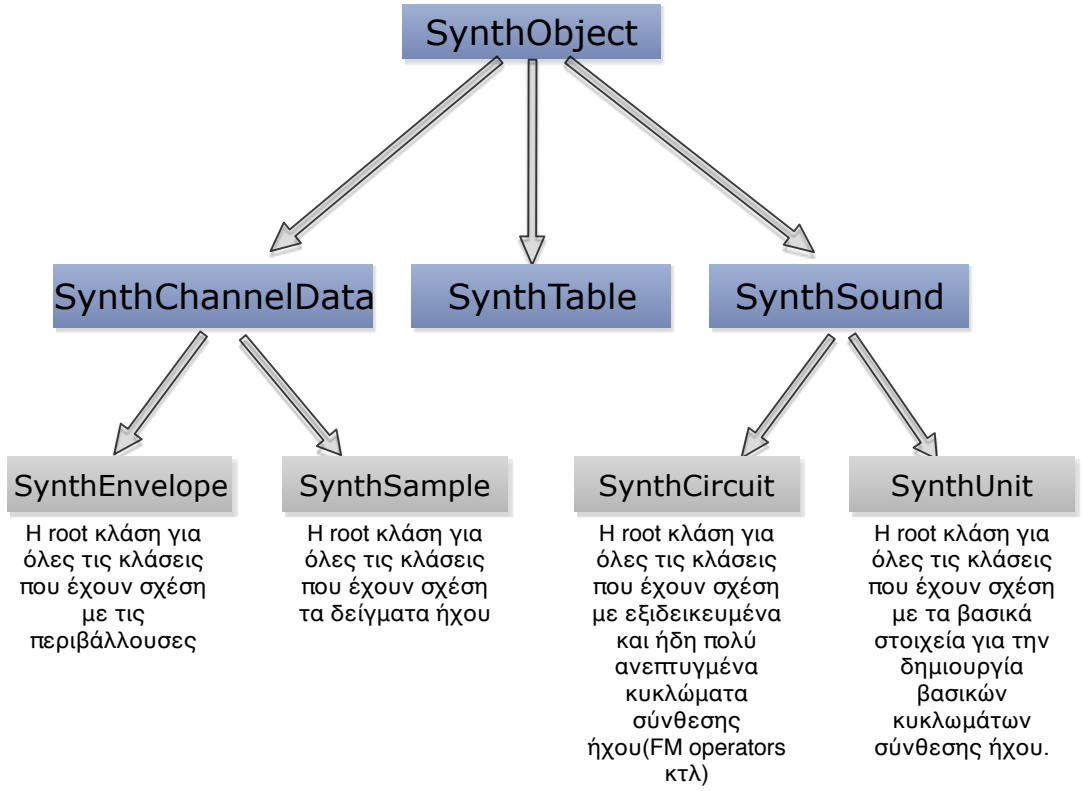
Παρακάτω ακολουθούν αναλυτικές οι περιγραφές των βασικών κλάσεων του JSyn.

Class SynthObject
Μητρική κλάση όλων των κλάσεων που συμμετέχουν στην δημιουργία εφαρμογών σύνθεσης ήχου. Η κλάση αυτή κρατάει στην μνήμη μια λίστα με όλα τα αντικείμενα που έχουν δημιουργηθεί στο πρόγραμμα σε περίπτωση απρόσμενης ενεργοποίησης του μηχανισμού garbage collection[3].
Δέντρο Κληρονομικότητας java.lang.Object | +--java.util.Observable | +--com.softsynth.jsyn.SynthObject

[3] Μηχανισμός της Java του οποίου στόχος είναι η απελευθέρωση της μνήμης του υπολογιστή από αντικείμενα που έχουν δημιουργηθεί κάποια στιγμή στην εφαρμογή και δεν χρησιμοποιούνται στην συνέχεια.

Μέθοδοι Constuctor SynthObject(SynthContext context)	
Return type	Μέθοδοι
void	delete() Σβήνει από την μνήμη αντικείμενα της κλάσης(αυτής ή κλάσεων που έχουν κληρονομήσει την μέθοδο αυτή)
static void	deleteAll() Σβήνει από την μνήμη όλα τα αντικείμενα της κλάσης.
static void	enableDeletionByGarbageCollector(boolean ifEnabled) Θέτει σε λειτουργία την διαγραφή αντικειμένων από τον μηχανισμό garbage collection.
static void	enableTracking(boolean ifTrack) Θέτει σε λειτουργία ή μη λειτουργία ένα μηχανισμό που εντοπίζει αυτόματα τα αντικείμενα που δημιουργούνται από το thread που λειτουργεί εκείνη τη στιγμή.
int	getPeer() Επιστρέφει τον αύξοντα αριθμό του Peer
SynthContext	getSynthContext() Επιστρέφει το αντικείμενο SynthContext στο οποίο είναι χτισμένη η εκάστοτε εφαρμογή
Static Boolean	isTrackingEnabled() Επιστρέφει την λογική τιμή true εάν έχουμε θέσει σε λειτουργία τον μηχανισμό Tracking από την παραπάνω μέθοδο enableTracking (boolean ifTrack)
void	setSynthContext(SynthContext context) Αντιστοιχεί ένα αντικείμενο synthContext για οποιοδήποτε αντικείμενο SynthObject

Ακολουθεί ένα διάγραμμα ιεραρχικής ακολουθίας της κλάσης synth object

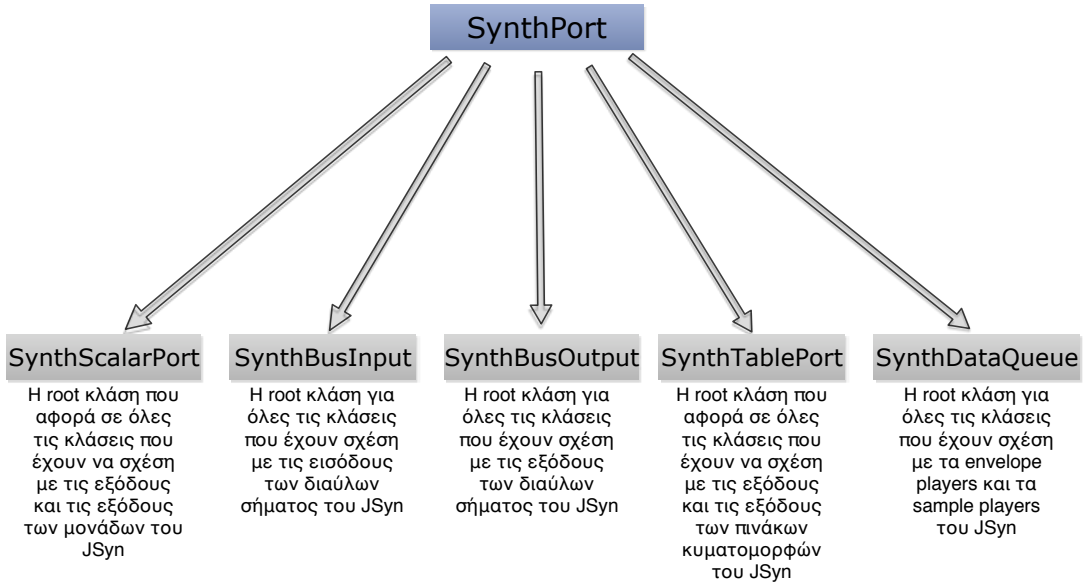


Class SynthPort
Μητρική κλάση όλων των κλάσεων που αφορούν τις συνδέσεις και τις δρομολογήσεις σημάτων μέσα στις μηχανές σύνθεσης ήχου που γίνονται με το JSyn. Χαρακτηριστικό της είναι ότι δεν έχει μέθοδο constructor κάτι που σημαίνει πως έχει φτιαχτεί μόνο για να κληρονομεί μεθόδους χωρίς να μπορεί να δημιουργήσει αντικείμενα.
Δέντρο Κληρονομικότητας java.lang.Object | +-- com.softsynth.jsyn.SynthPort

Return type	Μέθοδοι

java.lang.string	getAlias() Επιστρέφει το όνομα του εικονικού port του αντικειμένου.
java.lang.string	getName() Επιστρέφει το όνομα του port του αντικειμένου.
int	getNumParts() Επιστρέφει τον αριθμό των μελών ενός port.
int	getSignalType() Επιστρέφει τον τύπο του σήματος που περνάει από την port. Δηλαδή εάν το σήμα είναι signed(-1.0,+1.0), unsigned (0.0,2.0), εάν είναι ακουστική συχνότητα (Hz), συχνότητα δειγματοληψίας (Hz) ή τέλος αν είναι unsigned σήμα ημιζωής (half-life) ενός λογαριθμικού σήματος (0.0,1.0).
int	getSignalType(int PartIndex) Η ίδια μέθοδος με την παραπάνω με πρόσθετη δήλωση για πιο μέλος του port θέλουμε να μας επιστρέψει τον τύπο του σήματος.
SynthSound	getSound() Επιστρέφει το αντικείμενο SynthSound στο οποίο υπάρχει ο συγκεκριμένο port.
void	setAlias(java.lang.string string) Θέτει το εικονικό port από την οποία θα περάσει το σήμα.
void	setSignalType(int signalType) Ορίζει την port για να δέχεται ένα συγκεκριμένο τύπο σήματος από τα παρακάτω: Προσημασμένο (-1.0,+1.0), μη προσημασμένο (0.0,2.0), ακουστική συχνότητα (Hz), συχνότητα δειγματοληψίας (Hz), μη προσημασμένο σήμα ημιζωής ενός λογαριθμικού σήματος (0.0,1.0).
void	setSignalType(int signalType, int numPort) Ίδια μέθοδος με την παραπάνω με πρόσθετη την δήλωση σε πιο μέλος της port θέλουμε να ορίσουμε τον τύπο του σήματος.

Ακολουθεί ένα διάγραμμα ιεραρχικής ακολουθίας της κλάσης SynthPort



Class Synth	
Η κλάση Synth παρέχει γενικό έλεγχο στις σταθερές τιμές ενός synthesizer. Μοιράζει σε όλα τα αντικείμενα της εφαρμογής ένα πλαίσιο συγκεκριμένης συχνότητας δειγματοληψίας.	
Δέντρο Κληρονομικότητας java.lang.Object | +--com.softsynth.jsyn.Synth	
Return type	*Πεδία τιμών(Fields)*

static java.lang.String	copyright Επιστρέφει ένα string με τα πνευματικά δικαιώματα της εφαρμογής.
static int	FLAG_AUTO_STOP Χρησιμοποιείται μαζί με τις κλάσεις SynthEnvelopeQueue και SynthSampleQueue και τερματίζει την λειτουργία της εφαρμογής όταν σταματήσει η ροή δεδομένων (συνήθως σήματος) από την εφαρμογή.
static int	FLAG_DISABLE_OUTPUT Διακόπτει την ζωντανή έξοδο του σήματος κατά την ηχογράφηση όταν το σύστημα δεν υποστηρίζει full duplex ηχογράφηση.
static int	FLAG_NON_REAL_TIME Ζητάει την τέλεση μιας ηχητικής εργασίας να διεκπεραιωθεί όσο το δυνατόν ταχύτερα άλλες φορές γρηγορότερα και άλλες πιο αργά από την τέλεση της σε πραγματικό χρόνο.
static int	SIGNAL_TYPE_FULL_RANGE Δήλωση γενικού τύπου σήματος για τιμές μεγάλου εύρους =~-1 1.
static int	SIGNAL_TYPE_HALF_LIFE Δήλωση χρόνου ημιζωής του σήματος σε δευτερόλεπτα.
static int	SIGNAL_TYPE_OSC_FREQ Δήλωση της συχνότητας ενός ταλαντωτή με εύρος τιμών ±SR/2 Hz.
static int	SIGNAL_TYPE_RAW_SIGNED Δήλωση τύπου σήματος για μία μονάδα με εύρος -1.0,+1.0
static int	SIGNAL_TYPE_RAW_UNSIGNED Δήλωση τύπου σήματος για μία μονάδα με εύρος 0.0,2.0

static int	SIGNAL_TYPE_SVF_FREQ Δήλωση της συχνότητας ενός State Variable φίλτρου με εύρος τιμών ±SR/2 Hz.
static int	SIGNAL_TYPE_SAMPLE_RATE Δήλωση συχνότητας διαβάσματος ενός δείγματος (SampleReader) με εύρος [0.0, SR×2].
static int	SIGNAL_TYPE_TIME Δήλωση για την χρήση σήματος ώς τιμές χρόνου (sec).
static int	timeAdvance Μεταβλητή που χρησιμοποιείται για τον αλγοριθμικό έλεγχο του χρονοδιαγράμματος των γεγονότων που συμβαίνουν σε μία εφαρμογή.

Μέθοδοι Constuctor

Synth()

Return type	*Μέθοδοι*
java.lang.string	errorCodeToString() Επιστρέφει τον κωδικό τυχόν λάθους σε μορφή string.
static int	getFrameCount() Επιστρέφει τον αύξοντα αριθμό του τελευταίου δείγματος που υπολογίστηκε.
static double	getFrameRate() Επιστρέφει την συχνότητα δειγματοληψίας που υπάρχει εκείνη ακριβώς την στιγμή που καλείται. Έξ' ορισμου: 44100.0
static int	getTickCount() Επιστρέφει την ώρα σε tick.
static double	getTickRate() Επιστρέφει την συχνότητα επανάληψης των tick.

SynthContext	getSynthContext() Επιστρέφει το αντικείμενο SynthContext στο οποίο είναι βασισμένη η εκάστοτε εφαρμογή
Static Boolean	isTrackingEnabled() Επιστρέφει την λογική τιμή true εάν έχουμε θέσει σε λειτουργία τον μηχανισμό Tracking από την μέθοδο enableTracking (boolean ifTrack)
void	*setSynthContext*(SynthContext context) Αντιστοιχεί ένα αντικείμενο synthContext για οποιοδήποτε αντικείμενο SynthObject

Class SynthContext	
Η κλάση SynthContex παρέχει στον προγραμματιστή ένα πλαίσιο στο οποίο μπορεί να χτιστεί μια εφαρμογή σύνθεσης ήχου. Οι εφαρμογές που έχουν χτιστεί πάνω σε ένα μοναδικό πλαίσιο της κλάσης SynthContext μπορουν….Είναι απαραίτητη η χρήση της κλάσης στην ανάπτυξη applets.	
Δέντρο Κληρονομικότητας java.lang.Object | +--com.softsynth.jsyn.SynthContext	
SynthContext()	*Μέθοδοι Constuctor*
Return type	*Μέθοδοι*
void	deleteAll() Διαγράφει από τη μνήμη όλες τις μονάδες σύνθεσης που δημιουργήθηκαν από τη στιγμή που καλέστηκε η μέθοδος initialize().
void	enableDeletionByGarbageCollection() Θέτει σε λειτουργία τον μηχανισμό garbage collection.

int	getFrameCount() Επιστρέφει τον αύξοντα αριθμό της θέσης του πίνακα στην οποία είναι αποθηκευμένο το τελευταίο δείγμα που αναπαράχθηκε.
Double	getFrameRate() Επιστρέφει την συχνότητα δειγματοληψίας που χρησιμοποιεί η synthContext την στιγμή που καλείται.

Ανάλυση των κλάσεων της οικογένειας SynthObject

Οι σημαντικότερες κλάσεις του JSyn για τον καθαρό προγραμματισμό σύνθεσης ήχου είναι οι κλάσεις της οικογένειας SynthObject. Παρακάτω μπορεί κάποιος να δει αναλυτικά τις κλάσεις SynthChannelData, SynthSound και SynthTable οι οποίες είναι απευθείας παράγωγες της SynthObject. Επίσης θα αναλυθούν και οι κλάσεις SynthEnvelope, SynthSample (απευθείας παράγωγες της SynthChannelData), και τέλος SynthCircuit, SynthUnit(απευθείας παράγωγες της SynthSound).

Class SynthChannelData
Η κλάση SynthChannelData είναι ένας σταθμός αποθήκευσης για δεδομένα ψηφιακού ήχου. Τα πραγματικά δεδομένα αποθηκεύονται σε μνήμη που είναι πρόσβαση στο DMA[4] ή το DSP ώστε να είναι ανεξάρτητα από την κεντρική μονάδα επεξεργασίας. Επομένως όλες οι κλάσεις που έχουν κληρονομήσει την SynthChannelData δεν χρησιμοποιούν την μνήμη που δεσμεύεται από την εφαρμογή.
Δέντρο Κληρονομικότητας java.lang.Object | +--java.util.Observable |

[4] DMA(Direct Memory Access) είναι ένα σύστημα των μοντέρνων υπολογιστών που επιτρέπει συγκεκριμένα υλικά υποσυστήματα ενός υπολογιστή όπως κάρτα ήχου ή γραφικών να έχουν πρόσβαση στην κύρια μνήμη του υπολογιστή για διεργασίες που αφορούν τα ίδια τα υποσυστήματα μόνο, ανεξαρτήτως της CPU.

+--com.softsynth.jsyn.SynthObject

|

+--com.softsynth.jsyn.SynthObject.SynthChannelData

Μέθοδοι Constuctor

SynthChannelData()

SynthChannelData(int Saxifrages)

SynthChannelData(SynthContext context,int MaxFrames)

Return type	Μέθοδοι
CuePoint	findCuePoint(int uniqueID) Επιστρέφει το σημείο queue με την μοναδική ταυτότητα που ορίζεται μέσω ενός ακεραίου.
int	findCuePosition(int uniqueID) Επιστρέφει την θέση του σημείου queue σε ακέραιο αριθμό.
int	getMaxFrames() Επιστρέφει την μέγιστη τιμή του sample rate του αποθηκευμένου δείγματος.
void	setNumFrames() Θέτει νέο sample rate σε ένα αντικείμενο. Η τιμή αυτή ισχύει ώς τον αριθμό MaxFrames. Αν αυτή είναι μεγαλύτερη τότε αυτόματα ισχύει η τιμή MaxFrames.
int	getReleaseBegin() Επιστρέφει το σημείο όπου αρχίζει το release κομμάτι ενός envelope.
int	getReleaseEnd() Επιστρέφει το σημείο όπου τελειώνει το release ενός envelope
int	getReleaseSize() Επιστρέφει το μέγεθος του release ενός envelope
int	getSustainBegin() Επιστρέφει το σημείο όπου αρχίζει το sustain κομμάτι ενός envelope.

void	setReleaseLoop(int startFrame, int endFrame) Θέτει στο release κομμάτι του envelope τα σημεία (αρχή και τέλος) από όπου θα επαναλαμβάνεται.
int	getSustainEnd() Επιστρέφει το σημείο όπου τελειώνει το sustain κομμάτι ενός envelope.

Class SynthEnvelope

Η κλάση SynthEnvelope περιέχει ένα πλήθος ζευγάρια αριθμών διάρκειας-αξίας, τα οποία περιγράφουν μια συνάρτηση ή μια οποιαδήποτε διεργασία στον χρόνο. Μπορεί να χρησιμοποιηθεί για τον σχηματισμό περιβάλλουσας της έντασης των νοτών είτε για πολυποίκιλες διαμορφώσεις παραμέτρων.

Δέντρο Κληρονομικότητας

```
java.lang.Object
  |
  +--java.util.Observable
      |
      +--com.softsynth.jsyn.SynthObject
          |
          +--com.softsynth.jsyn.SynthObject.SynthChannelData
              |
              +--com.softsynth.jsyn.SynthObject.SynthChannelData.SynthEnvelope
```

Μέθοδοι Constuctor

SynthEnvelope(double[] data)

SynthChannelData(int MaxFrames)-Δημιουργία περιβάλλουσας με ορισμένο αριθμό frames.

SynthChannelData(SynthContext context,double[] data)- Δημιουργία περιβάλλουσας η οποία θα δημιουργηθεί από τα δεδομένα που περιέχονται στον δηλωμένο πίνακα data.

SynthChannelData(SynthContext context,int MaxFrames)

Return type	Μέθοδοι
void	read(double[] data)

	Διαβάζει απευθείας τον πίνακα που είναι αποθηκευμένος στο αντικείμενο
void	read(int firstEnvelopeFrame, double[] data, int firstDataFrame, int numFrames) Διαβάζει δεδομένα από τον πίνακα data σχετικά με την περιβάλλουσα.
void	write(double[] data) Γράφει έναν πίνακα δεδομένων στο αντικείμενο
void	write(int firstEnvelopeFrame, double[] data, int firstDataFrame, int numFrames) Μετατρέπει τα δεδομένα της περιβάλλουσας σύμφωνα με τα σημεία που δίνει ο πίνακας data.

Class SynthSample

Η κλάση SynthSample είναι ένας αποθήκευσης για δεδομένα ψηφιακού ήχου.

Δέντρο Κληρονομικότητας

```
java.lang.Object
  |
  +--java.util.Observable
      |
      +--com.softsynth.jsyn.SynthObject
          |
          +--com.softsynth.jsyn.SynthObject.SynthChannelData
              |
              +--com.softsynth.jsyn.SynthObject.SynthChannelData.SynthSample
```

Μέθοδοι Constuctor

SynthSample()

SynthSample(int numFrames) Δημιουργία ενός άδειου δείγματος με ορισμένο αριθμό frames.

SynthSample(int maxFrames, int channelsPerFrame)

SynthSample(SynthContext synthContext) Δημιουργία ενός άδειου δείγματος. SynthSample(SynthContext synthContext, int maxFrames) SynthSample(SynthContext synthContext, int maxFrames, int channelsPerFrame) Δημιουργία ενός άδειου δείγματος με ορισμένο αριθμό frames, και καναλιών ανά frame.	
Return Type	Πεδία
Static int	AIFF Τύπος αποθηκευμένων δεδομένων με data structure τύπου aiff
byte[]	byteData Χρησιμοποιείται για την προσωρινή αποθήκευση raw bytes όταν αναλύεται ένα αρχείο-δείγμα ήχου.
static int	UNRECOGNIZED Μη αναγνωρίσιμος τύπος δεδομένων
static int	WAV Τύπος αποθηκευμένων δεδομένων με data structure τύπου wav
Return type	Μέθοδοι
void	allocate(int numFrames, int channelsPerFrame) Δεσμεύει μνήμη και πόρους του υπολογιστή για να αποθηκευτούν τα δεδομένα του δείγματος.
void	clear() καθαρίζει το buffer
void	clear(int firstSampleFrame, int numFrames) Διαγράφει από…. μέρος του δείγματος.
java.lang.String	dump()
double	getBaseFrequency() Επιστρέφει την τιμή της συχνότητας που αντιλήφθηκε ώς την κύρια συχνότητα στην αναπαραγωγή του δείγματος στην κανονική του συχνότητα δειγματοληψίας.
int	getChannelsPerFrame()

	Επιστρέφει τον αριθμό των καναλιών ανά Frame.
long	getFileSize() Επιστρέφει τον αριθμό των bytes που πρέπει να διαβαστούν ώστε να ολοκληρωθεί η διαδικασία διαβάσματος.
static int	getFileType(java.lang.String fileName) Επιστρέφει τον τύπο αρχείου ήχου που ανήκει το δείγμα
double	getSampleRate() επιστρέφει την συχνότητα δειγματοληψίας
void	Load(java.io.InputStream stream) Φορτώνει ένα δείγμα από ένα συγκεκριμένο ρεύμα δεδομένων
void	load(java.io.InputStream stream, boolean ifLoadData) Φορτώνει ένα δείγμα από ένα συγκεκριμένο ρεύμα δεδομένων, με την επιλογή της πυροδότησης του φορτώματος
short[]	loadShorts(java.io.InputStream stream, boolean ifLoadData)
void	read(int firstSampleFrame, short[] data, int firstDataIndex, int numFrames) Διαβάζει τα αποθηκευμένα στο αντικείμενο δεδομένα με την επιλογή του σημείου που θα αρχίσει και την δειγματοληψία που θα το κάνει .
void	read(short[] data) Διαβάζει τα αποθηκευμένα στο αντικείμενο δεδομένα με τον προκαθορισμένο τρόπο.
void	setBaseFrequency(double baseFrequency) Θέτει την βασική συχνότητα σε ένα δείγμα
void	setHighAmplitude(double highestAmplitude) Θέτει το μέγιστο πλάτος έντασης στο δείγμα.
void	setHighFrequency(double highestFrequency) Θέτει την υψηλότερη συχνότητα του δείγματος
void	setLowAmplitude(double lowestAmplitude) Θέτει το ελάχιστο πλάτος έντασης στο δείγμα.
void	setLowFrequency(double lowestFrequency) Θέτει την χαμηλότερη συχνότητα του δείγματος
void	setSampleRate(double sampleRate) Θέτει το sample rate του δείγματος

void	write(int firstSampleFrame, short[] data, int firstDataIndex, int numFrames) Γράφει δεδομένα στον πίνακα του δείγματος με την επιλογή του πρώτου σημείου και του αριθμού των frames που θα γραφούνε.
void	write(short[] data) Γράφει δεδομένα στον πίνακα του δείγματος με τις προκαθορισμένες επιλογές

Class SynthSound

Χρησιμοποιείται για τα στοιχειώδη κυκλώματα και ηχητικές μονάδες.

Δέντρο Κληρονομικότητας

java.lang.Object

|

+--java.util.Observable

|

+--com.softsynth.jsyn.SynthObject

|

+--com.softsynth.jsyn.SynthObject.SynthSound

Μέθοδοι Constuctor

SynthSound()

SynthSound(SynthContext context)

Return type	Μέθοδοι
void	addPort(SynthPort port) Σχετίζει με ένα port
void	addPort(SynthPort port, java.lang.String alias) Προσθέτει μια θύρα στην μονάδα που περιέχει όλες τις θύρες για κάποιο συγκεκριμένο σήμα.
SynthPort	findNamedPort(java.lang.String name) Επιστρέφει το port με το συγκεκριμένο όνομα

java.lang.String	getName() Επιστρέφει το όνομα του port
int	getNumPorts() Επιστρέφει τον αριθμό των ports
SynthPort	getPortAt(int index) Επιστρέφει το port που βρίσκεται σε συγκεκριμένη θέση στον πίνακα
int	getPriority() Επιστρέφει το επίπεδο προτεραιότητας
void	setPriority(int priority) Θέτει το επίπεδο προτεραιότητας του port
void	start() Εκκινεί την εκτέλεση της εκάστοτε μονάδας
void	stop() Τερματίζει την εκτέλεση της εκάστοτε μονάδας

Class SynthCircuit

Τα SynthCircuit περιέχουν SynthUnits συνδεδεμένα μεταξύ τους έτσι ώστε να φτιάχνουν μεγαλύτερα κυκλώματα (patches).

Δέντρο Κληρονομικότητας

```
java.lang.Object
  |
  +--java.util.Observable
      |
      +--com.softsynth.jsyn.SynthObject
          |
          +--com.softsynth.jsyn.SynthObject.SynthSound
              |
              +--com.softsynth.jsyn.SynthObject.SynthSound.SynthCircuit
```

Μέθοδοι Constuctor
SynthCircuit() SynthCircuit(int numSounds) SynthCircuit(SynthContext synthContext) SynthCircuit(SynthContext synthContext, int numSounds)

Return type	Μέθοδοι
void	add(SynthSound sound) προσθέτει ένα συγκεκριμένο
void	compile() Ομαδοποιεί όλα τα αντικείμενα SynthSound που έχουν προστεθεί με την μέθοδο add() στο κύκλωμα.
void	delete() Διαγράφει όλες τις υπομονάδες και μετά το ίδιο το κύκλωμα.

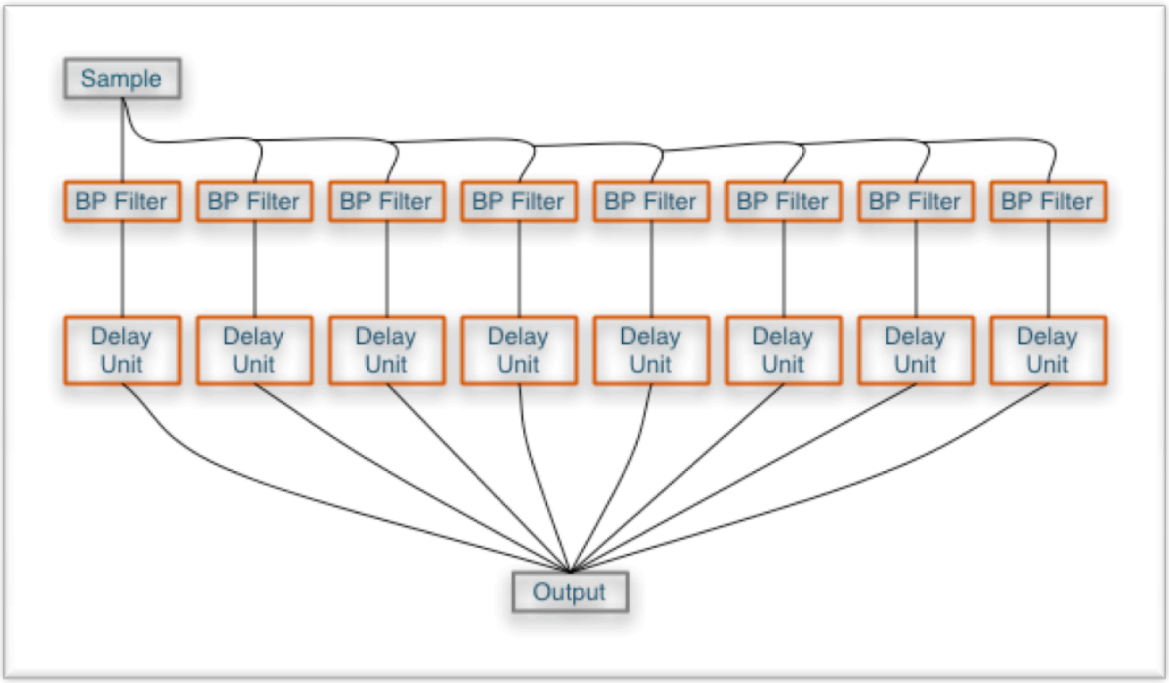
Κεφάλαιο 2: Οι εφαρμογές

2.1.Η εφαρμογή Despect

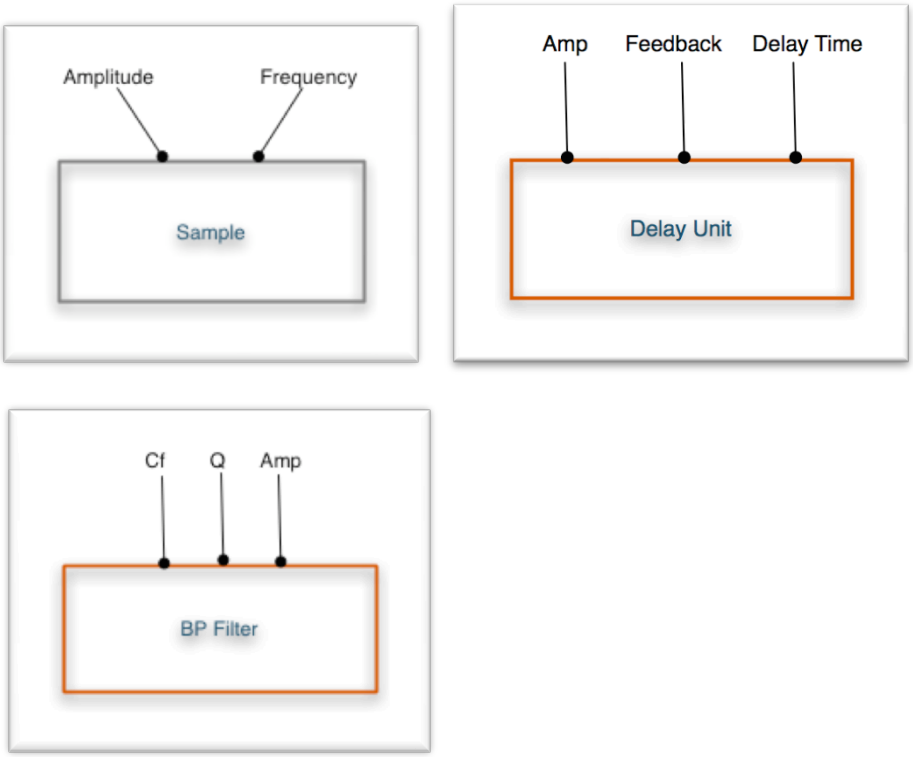
2.1.1 Περιγραφή της Εφαρμογής

Η εφαρμογή Despect σχεδιάστηκε έτσι ώστε να παρέχει σε ένα χρήστη οποιασδήποτε πλατφόρμας ηλεκτρονικού υπολογιστή την δυνατότητα για την δημιουργία ηχητικών μορφωμάτων και την επεξεργασία τους σε πραγματικό χρόνο μέσω γραφικού περιβάλλοντος (GUI).

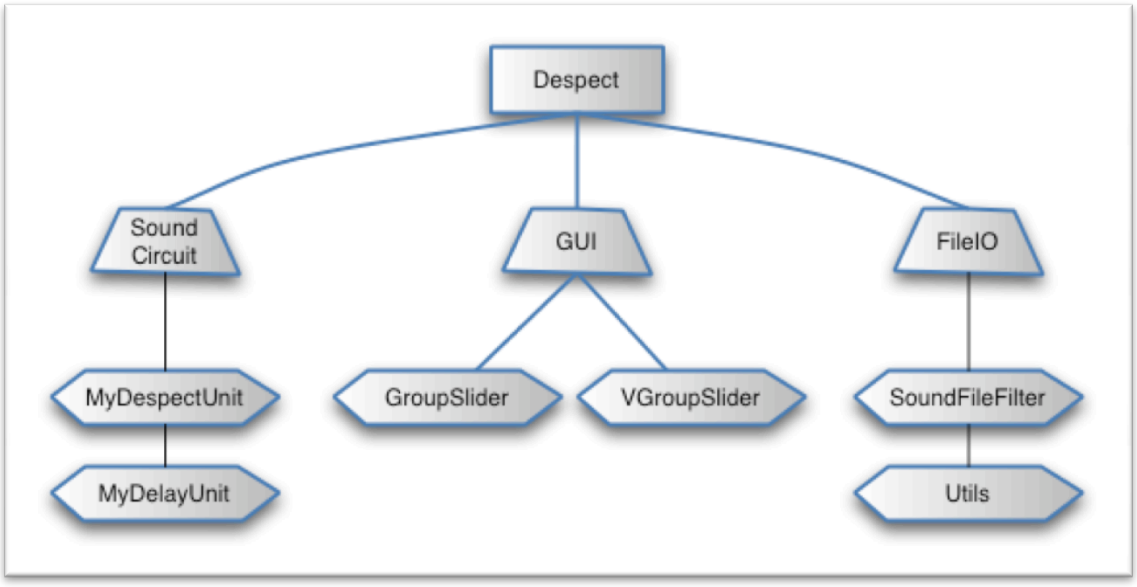
Ο σχεδιασμός της μηχανής ήχου προβλέπει την χρήση και τον συνδυασμό δυο τεχνικών σύνθεσης ήχου. Την αφαιρετική, την σύνθεση βασισμένη στην δειγματοληψία και την τεχνική με την χρήση σειρών καθυστέρησης. Πιο συγκεκριμένα η ηχητική πηγή η οποία στην συγκεκριμένη εφαρμογή επιλέχτηκε να είναι προηχογραφημένα αρχεία ήχου επιλογής του χρήστη συνδέεται παράλληλα με 10 φίλτρα διέλευσης ζώνης και το καθένα από αυτά με μια μονάδα καθυστέρησης. Τέλος προσθέτουμε τα σήματα των 10 αυτών μονάδων στην έξοδο μας.



Ο έλεγχος που έχει ο χρήστης στο ηχητικό αποτέλεσμα υλοποιείται μέσω ορισμένων sliders, τα οποία επηρεάζουν τρεις παραμέτρους σε κάθε φίλτρο, δύο παραμέτρους στις μονάδες καθυστέρησης και δυο παραμέτρους για τον έλεγχο του δείγματος ήχου. Ειδικότερα ο χρήστης μπορεί να ελέγχει την ένταση, την κεντρική συχνότητα και το Q του κάθε φίλτρου. Επίσης μπορεί να ελέγχει την ένταση και το ποσοστό ανατροφοδότησης της κάθε μονάδας καθυστέρησης. Τέλος ελέγχει την ένταση και την συχνότητα αναπαραγωγής του δείγματος. Οι χρόνοι καθυστέρησης δηλώνονται σαν σταθερές κατά την εκκίνηση της εφαρμογής και οι τιμές αυτές έχουν μπει με τρόπο ώστε να δημιουργείται ένα ενδιαφέρον αποτέλεσμα.



Η υλοποίηση της εφαρμογή Despect χωρίστηκε σε τρία στάδια. Στην δημιουργία του ηχητικού κυκλώματος, την δημιουργία γραφικού περιβάλλοντος και στην δημιουργία περιβάλλοντος για την διαχείριση των αρχείων. Στο κάθε ένα από τα τρία στάδια αυτά ήταν αναγκαία η δημιουργία εξωτερικών κλάσεων έτσι ώστε να γίνει σωστά η δόμηση της εφαρμογής και να αποκτηθεί μεγαλύτερος έλεγχος του τελικού αποτελέσματος σε σχέση με τον αρχικό σχεδιασμό.



2.1.2 Η Υλοποίηση του ηχητικού κυκλώματος

Το ηχητικό κύκλωμα της εφαρμογής σχεδιάστηκε εξ' ολοκλήρου με το πακέτο κλάσεων του JSyn API. Αρχικά κρίθηκε αναγκαία η δημιουργία καινούριων κλάσεων–ηχητικών μονάδων. Η root κλάση SynthCircuit του JSyn μας δίνει την δυνατότητα αυτή. Δηλώνοντας κατά την δημιουργία της δικής μας κλάσης ότι αυτή επεκτείνεται (δήλωση extends) στην κλάση SynthCircuit έχουμε αμέσως το δικαίωμα να χρησιμοποιήσουμε μεθόδους και πεδία της SynthCircuit. Έτσι μπορούμε να βάλουμε στην δικιά μας κλάση-ηχητική μονάδα άλλες ηχητικές μονάδες, εισόδους, εξόδους, και σημεία ελέγχου των παραμέτρων να τις συνδυάσουμε και να έχουμε την μονάδα που εμείς χρειαστήκαμε.

Αρχίζουμε με την κλάση που αφορά στην μονάδα με το delay.

```
public class MyDelayUnit extends SynthCircuit{…}
```

Αρχικά φτιάχνουμε ένα αντικείμενο της κλάσης InterpolatingDelayUnit. Αυτή μόνο η μονάδα του JSyn μας δίνει δικαίωμα να αλλάζουμε σε πραγματικό χρόνο τον χρόνο καθυστέρησης της κάθε μονάδας καθυστέρησης. Αυτό γίνεται θέτοντας κάθε φορά ένα καινούργιο χρόνο καθυστέρησης.

```
delayLine = new IntepolatingDelayUnit(delayTime)
```

Η μονάδα MyDelayUnit περιέχει μέσα άλλες δύο μονάδες πράξεων ηχητικού σήματος. Φτιάχνουμε λοιπόν αντικείμενα των κλάσεων-μονάδων αυτών και τα βάζουμε στο δικό μας κύκλωμα.

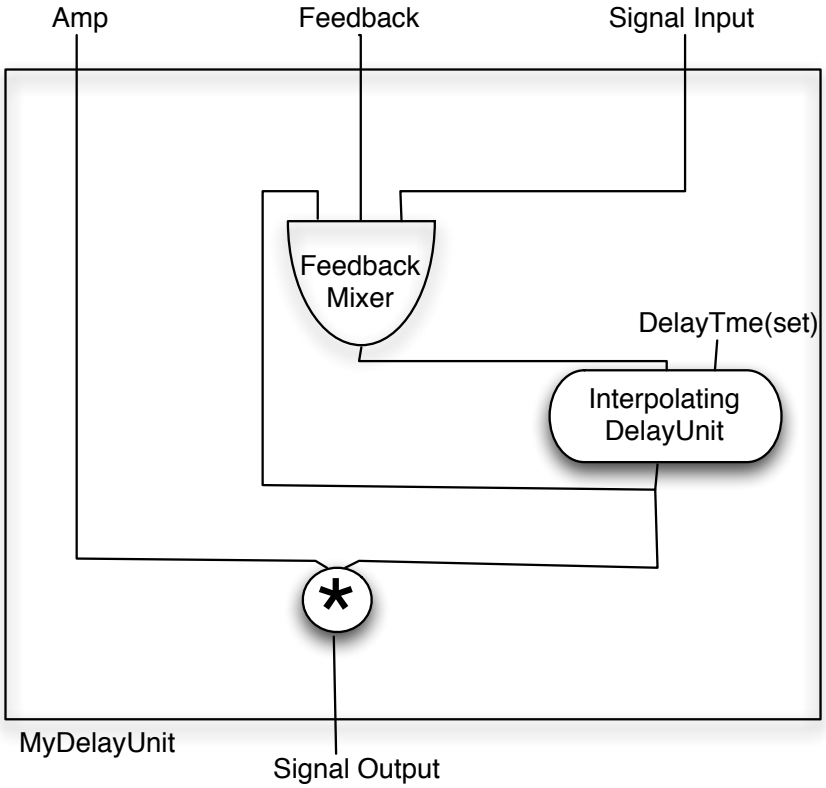
```
add( ampUnit = new MultiplyUnit );
add( feedbackMixer = new MultiplyAddUnit() );
```

Η μονάδα feedbackMixer είναι η μονάδα που μας προσθέτει το ποσοστό ανατροφοδότηση στο κύκλωμα, καθώς δίνει το αποτέλεσμα στην πράξη (α*β)+γ, όπου α,β,γ οι τρεις είσοδοι της. Έτσι βάζοντας στην είσοδο α το καθυστερημένο σήμα, στην είσοδο β τον αριθμό του ποσοστού ανατροφοδότησης που επιθυμούμε και στην γ το απευθείας σήμα που εισέρχεται στο κύκλωμα MyDelayUnit έχουμε το τελικό εφφε του delay. Η μονάδα ampUnit κάνει τον τελικό πολλαπλασιασμό για την ένταση της μονάδας

```
delayLine.output.connect( feedbackMixer.inputA );
feedbackMixer.output.connect( delayLine.input );
delayLine.output.connect(ampunit.inputB);
```

Τέλος κάνουμε τις κατάλληλες συνδέσεις, και προσθέτουμε στο κύκλωμα μας είσοδο, έξοδο και σημεία ελέγχου.

```
addPort( input = feedbackMixer.inputC, "input" );
addPort( feedback = feedbackMixer.inputB,"feedback" );
addPort( amplitude = ampunit.inputA );
addPort( output = ampunit.output );
```



Με τον ίδιο τρόπο φτιάχνουμε και την κλάση MyDespectUnit. Μια μονάδα στην οποία γίνονται οι συνδέσεις του MyDelayUnit και του φίλτρου διέλευσης ζώνης. Αρχικά δηλώνοντας την σύνταξη του Constructor της κλάσης MyDespectUnit δηλώνουμε σαν μεταβλητές τιμές τις παραμέτρους που θέλουμε να ελέγχουμε σε πραγματικό χρόνο από το γραφικό μας περιβάλλον.

```
public MyDespectUnit(double amp, double cfreq, double q,
double dlytime, double fdb, double delamp){…}
```

Έπειτα δημιουργούμε αντικείμενα των κλάσεων MyDelayUnit και Filter_BandPass και τα προσθέτουμε στο κύκλωμα.

```
add (filter = new Filter_BandPass());

add (dly = new MyDelayUnit(dlytime));
```

Ορίζουμε τι ακριβώς θέλουμε να ελέγξουμε από τα δυο αντικείμενα αυτά και τα συσχετίζουμε με τις μεταβλητές που ορίσαμε σαν παραμέτρους στον constructor της κλάσης.

```
filter.amplitude.set(amp);
filter.frequency.set(cfreq);
filter.Q.set(q);
dly.feedback.set(fdb);
dly.amplitude.set(delamp);
```

Μετά την δημιουργία του αντικειμένου filter και dly καλούμε τις μεθόδους και τα πεδία των κλάσεων τους. Στην συγκεκριμένη περίπτωση καλούμε όλα τα πεδία της κλάσης Filter_BandPass δηλαδή amplitude, frequency και Q τα οποία τα έχει κληρονομήσει από την μητέρα κλάση των φίλτρων com.softsynth.jsyn.TunableFilter.

Η μέθοδος set είναι ανήκει στην κλάση Synthnput. Με την μέθοδο set μπορούμε να δίνουμε τιμές στα πεδία amplitude, frequency και Q τα οποία έχουνε κληρονομηθεί από την κλάση TunableFilter.

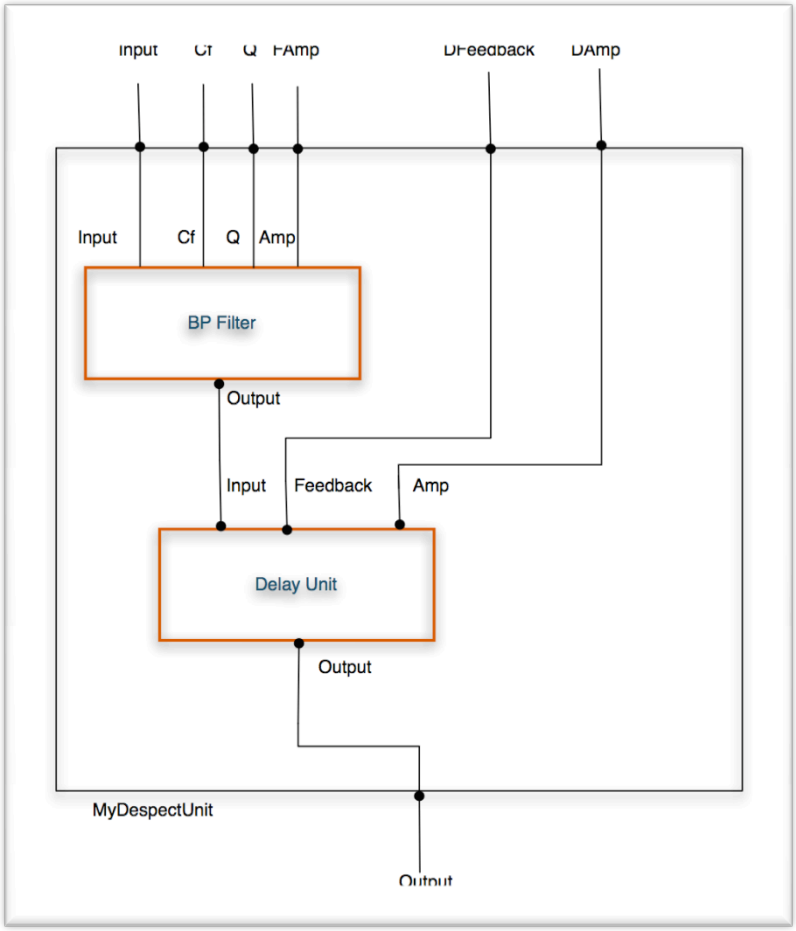
Με τον ίδιο τρόπο λειτουργούμε και για τα πεδία του αντικειμένου dly της κλάσης MyDelayUnit. Αυτήν τη φορά εμείς δηλώσαμε κατά την δημιουργία της κλάσης αυτής ότι το πεδίο feedback είναι τύπου SynthInput (**public** SynthInput feedback;).

Τώρα μπορούμε να κάνουμε τις συνδέσεις μας και να βάλουμε είσοδο και έξοδο για το σήμα. Με την addPort

```
filter.output.connect(0,dly.input,0);
addPort (input = filter.input,"input");
addPort (output = dly.output,"output");
```

Τέλος γράφουμε και δυο μεθόδους για να σταματάμε και να ξεκινάμε τα αντικείμενα της κλάσης MyDespectUnit όταν τις καλέσουμε στην κεντρική κλάση Enffected.

```
public void start(){
   filter.start();
    dly.start();
        }
public void stop(){
   filter.stop();
    dly.stop();
        }
```



Αυτό που απομένει να γίνει για να ολοκληρωθεί το ηχητικό μέρος της εφαρμογής είναι να δημιουργηθούν δέκα διαφορετικές εκδοχές της κλάσης MyDespectUnit, να εισαχθεί σήμα σε αυτές από συγκεκριμένο δείγμα ήχου, να προστεθούν οι έξοδοι τους και να μας δώσουν το τελικό αποτέλεσμα. Εφόσον αποφασίσαμε η ηχητική πηγή μας να είναι αρχεία προηχογραφημένου υλικού πρέπει να χρησιμοποιήσουμε μια κλάση του JSyn που μας δίνει την δυνατότητα να το κάνουμε. Αυτή είναι η κλάση SynthSample και την δηλώνουμε στο μπλοκ του κώδικά με τις αρχικές δηλώσεις ώς:

```
public SynthSample      mySamp;
```

Υπάρχουν όμως δύο διαδεδομένοι τύποι αρχείων, Τα αρχεία wav και τα αρχεία aiff. Αν φορτώναμε σε ένα buffer οποιοδήποτε τύπο από τους δύο θα δημιουργούταν πρόβλημα καθώς το κάθε αρχείο πρέπει να διαβαστεί με συγκεκριμένο τρόπο. Η κλάση SynthSample μας δίνει την ευκαιρία να χρησιμοποιήσουμε και τους δυο τύπους αυτούς καθώς έχει δυο υποκλάσεις που προβλέπουν η καθεμιά ξεχωριστό τύπο αρχείου. Έτσι έχοντας ανοίξει το αρχείο με μια λειτουργία την οποία θα δούμε αναλυτικά παρακάτω διαχωρίζουμε τις περιπτώσεις και προχωράμε στην φόρτωση του αρχείου στον buffer που στην προκειμένη περίπτωση θα είναι μία από τις δυο κλάσεις SynthSampleAIFF και SynthSampleWAV. Πρέπει να σημειωθεί ότι αυτός ο διαχωρισμός είναι εφικτός με την χρήση της μεθόδου getFileType(file) την οποία παρέχει η κλάση SynthSample. Αν ο χρήστης προσπαθήσει να φορτώσει κάποιο άλλο τύπο αρχείου αυτό δεν θα φορτωθεί και θα εμφανιστεί μήνυμα λάθους στην

κονσόλα. Μόλις βεβαιωθεί ο παρακάτω μηχανισμός για την εγκυρότητα του αρχείου θα το φορτώσει στην προσωρινή μονάδα μνήμης το αρχείο μέσω ενός ορισμένου ρεύματος ροής δεδομένων (stream).

```
switch( SynthSample.getFileType(openedFileName) ){
     case SynthSample.AIFF:
     mySamp = new SynthSampleAIFF();
     break;
     case SynthSample.WAV:
     mySamp = new SynthSampleWAV();
     break;
     default:SynthAlert.showError(this,
     "Unrecognized sample   filetype.   Please open only 16bit
     mono .aiff or .wav sound files");
     break;
     }
     if( mySamp != null ) loadSample( mySamp, stream );
```

Τέλος φτιάχνουμε πίνακες με τα αντικείμενα που χρειαζόμαστε και τα συνδέουμε ώστε να ολοκληρωθεί το κύκλωμα. Θα χρειαστούν στο κύκλωμα και οι κλάσεις του JSyn BusWriter και BusReader για μεγαλύτερη ευκολία στις συνδέσεις καθώς έχουμε μεγάλο αριθμό αντικειμένων που πρέπει να συνδεθούν στην έξοδο. Οι δίαυλοι σήματος, όπως περιγράφονται και στο δεύτερο κεφάλαιο, BusReaders και BusWriters χρησιμοποιούνται για την μίξη σημάτων και την αποφυγή χρήσης πολλών άλλων αντικειμένων που θα έκανε την εφαρμογή πιο βαριά. Επίσης χρησιμοποιήθηκε και η κλάση SynthDistributor η οποία μπορεί μοιράζει ένα σήμα σε πολλές ηχητικές μονάδες και διασφαλίζει ότι πάντα το σήμα είναι ίδιου τύπου. Στην συγκεκριμένη περίπτωση θέλουμε το σήμα να είναι πάντα full range (από -1 ώς 1) και να εισαχθεί ταυτόχρονα σε δέκα αντικείμενα MyDespectUnit . Ακολουθούμε λοιπόν τα παρακάτω βήματα:

Πρώτα αρχικοποιούμε:

```
MyDespectUnit[]   spect;

BusWriter[] mixin;
BusReader   mixout;
```

Αμέσως μετά δημιουργούμε τα δέκα MyDespectUnits

```
for(int i=0;i<10;i++){
     spect[i] = new MyDespectUnit(fampArray[i], FCfArrayinit[i],
FQArrayOut[i], DtArrayOut[i], DFdbArrayOut[i], DAmpArrayOut[i]);
}
```

Στις δηλώσεις για την δημιουργία των αντικειμένων έχουμε τους μεταβλητούς πίνακες οι οποίοι γεμίζουν τιμές από τις αρχικές θέσεις των sliders της εφαρμογής και θα εξεταστούνε παρακάτω αναλυτικά.

Τώρα γίνονται οι κατάλληλες συνδέσεις:

```
mixin = new BusWriter[10];
mixout = new BusReader();
```

```
SynthDistributor distributor = new
SynthDistributor("divideSample",Synth.SIGNAL_TYPE_FULL_RANGE);


distributor.connect(mySampler.output,0);

for (int i=0;i<10;i++){distributor.connect(spect[i].input, 0);
                        mixin[i] = new BusWriter();
                        spect[i].output.connect(0,mixin[i].input,0);
                    mixin[i].busOutput.connect(0,mixout.busInput,0);
                    spect[i].start();
                    mixin[i].start();
                    System.out.println("Unit #"+i+" connected");
                        }

. . .

mixout.output.connect(0,myOut.input,0);
mixout.output.connect(0,myOut.input,1);
```

Το σήμα της αναπαραγωγής του δείγματος ήχου συνδέεται στο αντικείμενο distributor. Αυτός με την σειρά του συνδέεται στις εισόδους των αντικειμένων spect της κλάσης MyDespectUnit. Οι δέκα έξοδοι των αντικειμένων αυτών εισέρχονται στα αντικείμενα mixin της κλάσης BusWriter, μιξάρονται στο αντικείμενο mixout της κλάσης BusReader, και το τελικό σήμα συνδέεται στην έξοδο.

2.1.3 Η Υλοποίηση του Graphical User Interface

Η κεντρική ιδέα της δημιουργίας του GUI κάθε εφαρμογής είναι να γίνει εύχρηστη και άμεση για τον κάθε χρήστη αλλά να παρέχει και ακρίβεια κατά τον έλεγχο της. Στην συγκεκριμένη εφαρμογή λόγω του πλήθους των παραμέτρων που μπορούν να επέμβουν στο τελικό ηχητικό αποτέλεσμα έγινε προσπάθεια να διατηρηθούν μόνο οι απαραίτητες παράμετροι και επιπλέον να είναι εφικτός και ο μαζικός έλεγχος παραμέτρων ώστε εκτός από τον λεπτομερή έλεγχο να έχουμε και δραστικό έλεγχο του τελικού αποτελέσματος. Αποφασίστηκε λοιπόν να αναπτυχθεί μια κλάση με την οποία θα ομαδοποιούνται κατά δεκάδες οι ελεγχτές των παραμέτρων, όσα άλλωστε και τα αντικείμενα MyDespectUnit τα οποία είναι φτιαγμένα να φιλτράρουν διαφορετικά εύρη συχνοτήτων με διαφορετικό Q και διαφορετικό χρόνο καθυστέρησης και ένταση. Αποφασίστηκε επίσης να χρησιμοποιηθούν για ελεγκτές τα sliders που παρέχει η Java με την κλάση JSlider που ανήκει στο πακέτο Javax.swing. Επίσης ένα κουμπί JButton το οποίο θα έδινε μια τυχαία τιμή σε κάθε ένα από τα δέκα sliders έτσι ώστε να γίνεται μια δραστική αλλαγή στο τελικό ηχητικό αποτέλεσμα, καθώς θα επηρέαζε όλο το φάσμα. Βέβαια η κλάση αυτή έπρεπε να γίνει γενικής χρήσης για πιθανή χρήση της στο μέλλον. Το μόνο πρόβλημα για να γίνει αυτό ήταν ότι για κάποιες χρήσεις όπως οι εντάσεις, οι μέγιστες και οι ελάχιστες τιμές είναι ίδιες- για παράδειγμα από 0 μέχρι 1 σε όλα τα sliders- ενώ για τιμές συχνοτήτων χρειαζόμαστε διάφορες τιμές για την περιγραφή όλου του φάσματος με ακρίβεια. Για παράδειγμα το πρώτο slider για τιμές κεντρικής συχνότητας χρειάστηκε να έχει εύρος από 20 Hz μέχρι 80 Hz, το δεύτερο από 80 μέχρι 180 κτλ. Για την πρώτη περίπτωση αρκεί η δήλωση κατά την

δημιουργία του αντικειμένου μέγιστης και ελάχιστης τιμής. Έτσι όλα τα sliders παίρνουν αυτές τις τιμές. Στην δεύτερη περίπτωση θα πρέπει να δηλωθούν πίνακες τιμών για να αντιστοιχηθεί για κάθε slider μια τιμή στα πεδία μέγιστη, ελάχιστη και αρχική τιμή. Αποφασίστηκε λοιπόν να γίνει μια κλάση για κάθε περίπτωση. Για την πρώτη περίπτωση η κλάση ονομάστηκε GroupSlider.

Στις αρχικές δηλώσεις της κλάσης αυτής δηλώθηκε ότι είναι προέκταση της κλάσης JPanel, για να έχουμε δικαίωμα χρήσης κάποιων κλάσεων και προσθήκης κατευθείαν στην κλάση.

```
public class GroupSlider extends JPanel{
```

Στον constructor του αντικειμένου πρέπει να δηλωθούν οι παράμετροι για την σωστή δημιουργία και χρήση αυτής της κλάσης:

```
public GroupSlider(String purpose, int numSliders, double min, double
max, double init){
```

Όταν χρησιμοποιείται η κλάση αυτή θα πρέπει να δηλώνονται με την σειρά ένα όνομα για την χρήση της κλάσης, ώστε να εμφανίζεται σαν επικεφαλίδα και να βοηθάει τον χρήστη στο τι επηρεάζει κάθε φορά, το πλήθος των sliders, την ελάχιστη τιμή, την μέγιστη τιμή και τέλος την αρχική τιμή που θα έχουν τα sliders.

Τώρα πρέπει να δηλωθεί το Layout για το panel αυτό ώστε όταν προστεθούν τα αντικείμενα με τάξη και με τρόπο που θα βοηθάει την διάδραση χρήστη-εφαρμογής

```
this.setLayout(new BorderLayout());
this.setBorder(new TitledBorder(new EtchedBorder(),purpose));
JPanel panelup = new JPanel(new GridLayout(0,num));
JPanel panelmain = new JPanel(new GridLayout(0,num));
paneldown = new JPanel(new BorderLayout());
```

Αποφασίστηκε η δημιουργία τριών καινούργιων panels και η προσθήκη τους με τον BorderLayout. Κάθε νέο panel θα δεχθεί και ένα αντικείμενο ή ομάδα αντικειμένων με ξεχωριστή λειτουργία. Αυτά θα χωριστούν από τον BorderLayoutManager στο επάνω, στο κάτω μέρος, και στη μέση του μητρικού panel με τον παρακάτω τρόπο.

```
this.add(panelup,BorderLayout.NORTH);
this.add(panelmain,BorderLayout.CENTER);
this.add(paneldown,BorderLayout.SOUTH);
```

Αμέσως μετά δημιουργούμε τα sliders και τα labels του καθενός GroupSlider και τα προσθέτουμε στο κεντρικό και το “βόρειο” panel αντίστοιχα.

```
for(int i=0;i<num;i++){
                JLabel initlabel = new JLabel("f" + (i+1));
                panelup.add(initlabel);}
for(int i=0;i<num;i++){
                cfslider[i] = new
                JSlider(slidermin,slidermax,sliderinit);
                cfslider[i].setOrientation(1);
```

```
                cfslider[i].setBackground(new Color(20));
                panelmain.add(cfslider[i]);}
```

Πρέπει να σημειωθεί πως η μεταβλητή num παίρνει την τιμή της από την αρχική δήλωση στον constructor για το πλήθος των sliders.

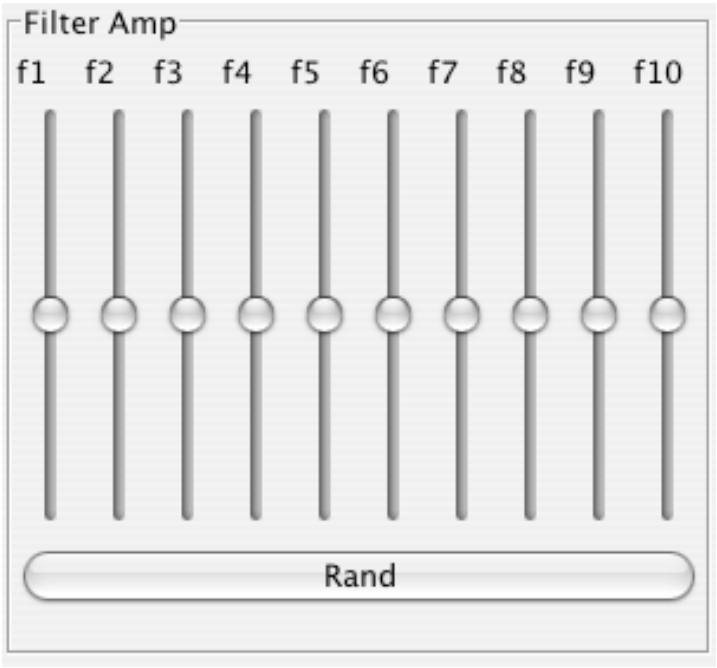
Τέλος φτιάχνουμε το κουμπί για την τυχαία αλλαγή τιμών του κάθε ενός slider.

```
randButton = new JButton("Rand");

paneldown.add(randButton,BorderLayout.PAGE_START);
```

Εξασφαλίζουμε και την λειτουργία του καθώς είναι πιο βολικό να παίρνουμε τις τιμές πάντα μέσα από το αντικείμενο αυτό. Είτε από την κίνηση των sliders είτε από το πάτημα του κουμπιού Rand. Έτσι φτιάχνουμε ένα αντικείμενο της κλάσης Random και εξασφαλίζουμε ότι λειτουργεί με τις τιμές που δεχθήκαμε κατά την δημιουργία του αντικειμένου, καθώς η κλάση random λειτουργεί μόνο με ακέραιους αριθμούς.

```
rand=new Random(888);
for(int i=0;i<num;i++){ArrayOutValues[i] = init;}

slidermin = (int)(min*1000);
slidermax = (int)(1000*max);
sliderinit = (int)(init*1000);
```

Τελικά θα έχουμε:



Στην περίπτωση του GroupSlider που αφορά τη ρύθμιση των συχνοτήτων το μόνο που πρέπει να αλλάξουμε είναι η δήλωση του constructor. Έτσι λοιπόν θα έχουμε:

```
public VariableGroupSlider(String purpose, int numSliders, int[]
minValues, int[] maxValues, int[] initValues){
```

Θα πρέπει να σημειωθεί ότι οι πίνακες περιμένουν πλήθος τιμών ίσο με τον ακέραιο numSliders που αντιπροσωπεύει το πλήθος των sliders που θα σχηματιστούν.

Μας μένει να τοποθετήσουμε τα sliders στο κεντρικό παράθυρο της εφαρμογής το οποίο δημιουργείται στην main κλάση Enffected.

```
public class Enffected extends JPanel implements
ActionListener, ChangeListener {
```

Δημιουργούμε λοιπόν τα αντικείμενα που θέλουμε για κάθε παράμετρο ξεχωριστά.

```
filtAmpGS = new GroupSlider("Filter Amp",10,0,1,0.5);
                  dlyTimeGS = new GroupSlider("Delay
                  Times",10,0.001,1.000,0.500);
filtCfGS = new VariableGroupSlider("Filter
CenterFr",10,FCfArraymin,FCfArraymax,FCfArrayinit);
dlyFdbGS = new GroupSlider("Feedback",10,0.0,1.0,0.9);
filtQGS = new GroupSlider("Filter Q",10,0,100,50);
dlyAmpGS = new GroupSlider("Delay Amps",10,0.0,1.0,0.95);
```

Επίσης δημιουργούμε και τα αντικείμενα για τα γραφικά γενικού ελέγχου. Αυτά που θα μας χρειαστούν είναι 2 sliders, το ένα για τον έλεγχο της γενικής έντασης του δείγματος και το άλλο για τον έλεγχο της συχνότητας του δείγματος, και δυο κουμπιά το ένα για την ενεργοποίηση του filechooser και το άλλο για την έναρξη της αναπαραγωγής. Επίσης δημιουργούμε και το γραφικό αντικείμενο για τον filechooser τον οποίο θα δούμε πιο αναλυτικά στην επόμενη παράγραφο.

```
                  openButton = new JButton("Open A Sound File");
                  hitButton = new JButton("Hit!");
                  fc=new JFileChooser();
```

Για να τα τοποθετήσουμε στο κεντρικό παράθυρο της εφαρμογής πρέπει να δημιουργήσουμε τις κατάλληλες συνθήκες ώστε να παρουσιαστούν με έναν τρόπο που θα προσφέρει ευχρηστία. Επειδή θα χρειαστούμε επιπλέον γραφικά στοιχεία για τον γενικό έλεγχο της εφαρμογής πρέπει να χωρίσουμε το κεντρικό παράθυρο σε 2 μέρη. Το ένα θα έχει στο κέντρο όλα τα γραφικά αντικείμενα GroupSlider και το άλλο τα στοιχεία γενικού ελέγχου. Έχουμε κάνει δήλωση στις αρχικές δηλώσεις της κλάσης ότι αυτή επεκτείνεται στην κλάση JPanel. Έτσι γράφοντας:

```
super(new BorderLayout());
```

θέτουμε στο κεντρικό παράθυρο τον BorderLayout Manager. Στο κέντρο τοποθετούμε ένα panel το οποίο θα δεχτεί τα GroupSliders και στο κάτω μέρος ένα panel το οποίο θα δεχθεί τα γραφικά γενικού ελέγχου. Προσθέτουμε τα panels στο κεντρικό παράθυρο, θέτουμε για αυτά έναν νέο layout Manager και προσθέτουμε σε

αυτά τα γραφικά αντικείμενα ελέγχου.

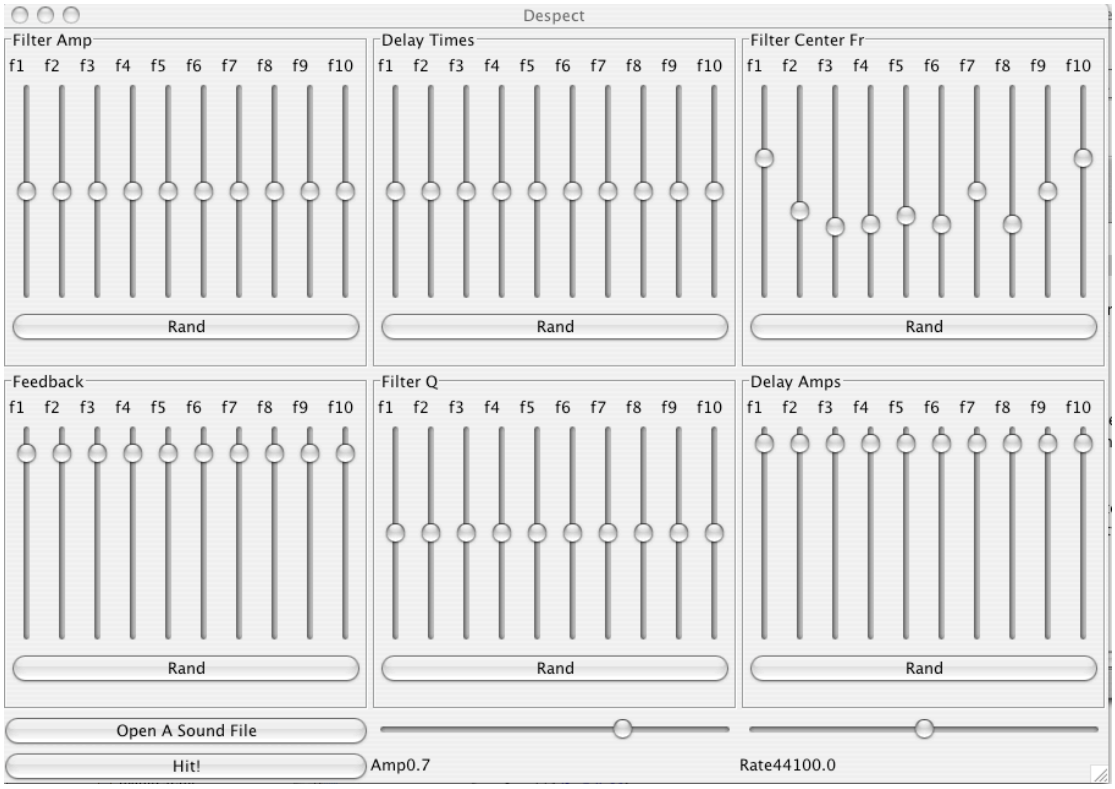
```
buttonPanel = new JPanel(new GridLayout(2,4));
gAmp = new JLabel("Amp" + genAmpValue);
gRate = new JLabel("Rate" + initRateValue);
JPanel cPanel = new JPanel(new GridLayout(2,3));
cPanel.add(filtAmpGS);
cPanel.add(dlyTimeGS);
cPanel.add(filtCfGS);
cPanel.add(dlyFdbGS);
cPanel.add(filtQGS);
cPanel.add(dlyAmpGS);
buttonPanel.add(openButton);
buttonPanel.add( genAmpSlider );
buttonPanel.add( genRateSlider );
buttonPanel.add(hitButton);
buttonPanel.add( gAmp );
buttonPanel.add( gRate);
add(cPanel, BorderLayout.CENTER);
add(buttonPanel, BorderLayout.PAGE_END);
```

Για να εμφανιστεί τελικά το κεντρικό παράθυρο φτιάχνουμε μια μέθοδο που θα την καλέσουμε μέσα στην main μέθοδο. Την ονομάζουμε createAndShowGUI:

```
public static void createAndShowGUI() {

            JFrame.setDefaultLookAndFeelDecorated(true);
            JFrame frame = new JFrame("Despect");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            JComponent newContentPane = new Enffected();
            newContentPane.setOpaque(true);
            frame.setContentPane(newContentPane);
            frame.pack();
            frame.setVisible(true);
        }
```

Μέσα σ'αυτήν την μέθοδο φτίαχνουμε ένα καινούργιο αντικείμενο της κλάσης JFrame. Το αντικείμενο της είναι το κεντρικό παράθυρο μέσα στο οποίο φιλοξενούνται όλα τα γραφικά που φτιάξαμε ώς τώρα. Πρέπει να δηλωθεί ορατό και να μπουν σ'αυτό γραφικά όρια ώστε να είναι ένα γραφικά σαφές παράθυρο. Τα όρια μας τα δίνει το αντικείμενο contentpane. Επίσης ορίζουμε κάποιες λεπτομέρειες της επιλογής μας όπως για το πώς θα αντιδρά η εφαρμογή στο πάτημα του κουμπιού για το κλείσιμο της και το πώς θα διακοσμηθεί.

Η τελική μορφή της εφαρμογής θα είναι η εξής:



Εφόσον τα γραφικά είναι έτοιμα πρέπει να συνδεθούν με την μηχανή ήχου ώστε ο χρήστης να ελέγχει τις παραμέτρους της. Για τον σκοπό αυτό πρέπει να χρησιμοποιηθούν οι Listeners. Προσθέτουμε λοιπόν έναν listener σε κάθε ένα από τα controls. Πρέπει να σημειωθεί ότι κάθε είδος ελεγκτή χρησιμοποιεί και έναν ορισμένο listener. Στην εφαρμογή αυτή χρησιμοποιούνται τα JSliders και τα JButtons. Αυτά χρειάζονται Change και Action Listeners αντίστοιχα. Έτσι γράφουμε στην κεντρική κλάση:

```
openButton.addActionListener(this);
hitButton.addActionListener(this);
genAmpSlider.addChangeListener(this);
genRateSlider.addChangeListener(this);
for (int i=0;i<10;i++){
                filtAmpGS.cfslider[i].addChangeListener(this);
                dlyTimeGS.cfslider[i].addChangeListener(this);
                filtCfGS.cfslider[i].addChangeListener(this);
                dlyFdbGS.cfslider[i].addChangeListener(this);
                filtQGS.cfslider[i].addChangeListener(this);
                dlyAmpGS.cfslider[i].addChangeListener(this);
}
                filtAmpGS.randButton.addActionListener(this);
                dlyTimeGS.randButton.addActionListener(this);
                filtCfGS.randButton.addActionListener(this);
                dlyFdbGS.randButton.addActionListener(this);
```

```
filtQGS.randButton.addActionListener(this);
dlyAmpGS.randButton.addActionListener(this);
```

Τώρα πρέπει να δημιουργηθούν οι μέθοδοι actionPerformed και stateChanged μέσα στις οποίες μπορούμε να γράψουμε τις ενέργειες που θέλουμε να γίνουν όταν ο χρήστης διαδρά με τα γραφικά ελέγχου.

```
public void actionPerformed(ActionEvent e) {…}

public void stateChanged(ChangeEvent e) {…}
```

Αυτές οι μέθοδοι είναι συγκεκριμένες για την εφαρμογή των Listeners και επιστρέφουν συγκεκριμένα αντικείμενα αναλόγως την χρήση τους.

Η λειτουργία των κουμπιών rand είναι η εξής. Όταν πατιέται το κουμπί αμέσως όλα τα sliders του group θα παίρνουν μια καινούργια τυχαία τιμή. Η λειτουργία αυτή περιγράφεται με τον εξής κώδικα μέσα στην μέθοδο actionPerformed:

```
for(int i=0;i<10;i++){
      if (e.getSource() == filtAmpGS.randButton) {
            int r=filtAmpGS.rand.nextInt(filtAmpGS.slidermax-
            filtAmpGS.slidermin)+filtAmpGS.slidermin;
            filtAmpGS.cfslider[i].setValue(r);
            filtAmpGS.cfslider[i].getValue();
            spect[i].filter.amplitude.set(po/1000);
            }
}
```

Με λίγα λόγια η λειτουργία του κώδικα αυτού είναι: Εάν η πηγή του αντικειμένου της κλάσης ActionEvent e είναι το JButton randButton που βρίσκεται στο GroupSlider filtAmpGS τότε δημιουργεί έναν νέο τυχαίο αριθμό του οποίου η τιμή θα είναι μέσα στο εύρος της ελάχιστης και της μέγιστης τιμής του slider. Θέτει κάθε μια φορά από τις δέκα (for(int i=0; i<10;i++)) έναν νέο αριθμό σε κάθε ένα slider του GroupSlider filtAmpGS και επιπλέον θέτει τις τιμές αυτές στο πλάτος των δέκα φίλτρων του MyDespectUnit spect. Με τον ίδιο τρόπο λειτουργούν και οι υπόλοιποι listeners των JButtons.

Η λειτουργία των listeners των JSliders είναι η εξής: Όταν το αντικείμενο e της κλάσης ChangeEvent ενεργοποιηθεί λόγω αλλαγής από κάποιο slider, αλλάζει ταυτόχρονα την τιμή της παραμέτρου στην μηχανή ήχου. Ένα παράδειγμα είναι το παρακάτω με τα sliders για τον έλεγχο των εντάσεων των φίλτρων:

```
for(int i=0;i<10;i++){
    if(e.getSource() == filtAmpGS.cfslider[i]){
                        JSlider source = (JSlider) e.getSource();
        if (source.getValueIsAdjusting()) {
double initSlideValue = (double)(filtAmpGS.cfslider[i].getValue());
double realValue = initSlideValue/1000;
fampArray[i] = realValue;
filtAmpGS.labeldown.setVisible(true);
filtAmpGS.labeldown.setText(filtAmpGS.pur+(i+1)+ "--->"+
```

```
String.valueOf(fampArray[i]));
spect[i].filter.amplitude.set(fampArray[i]);
}else{filtAmpGS.labeldown.setText(" ");
            }
      }
}
```

2.1.4 Η Υλοποίηση του file I/O

Ένα ακόμα σημαντικό κομμάτι της εφαρμογής είναι η λειτουργία της επιλογής των αρχείων ήχου από τον χρήστη ώστε να παραχθεί το ηχητικό αποτέλεσμα.

Για την ευχρηστία της λειτουργίας αυτής αποφασίστηκε η χρήση της κλάσης JFileChooser. Η κλάση αυτή, η οποία βρίσκεται στο πακέτο της Java javax.swing είναι ένα γραφικό παράθυρο το οποίο επιτρέπει την πλοήγηση του χρήστη σε όλο το δέντρο φακέλων και αρχείων του συστήματος στο οποίο τρέχει η εφαρμογή. Το αντικείμενο της κλάσης JFileChooser έχει την ιδιότητα να επιστρέφει το αρχείο που επιλέχθηκε ώς αντικείμενο της κλάσης File. Αυτό γίνεται με την χρήση της μεθόδου getSelectedFile() της κλάσης JFileChooser.

Στην εφαρμογή Despect χρειάστηκε εκτός από την παραπάνω ιδιότητα, δηλαδή η επιστροφή του αρχείου, για το φόρτωμά του στο stream που χρησιμοποιεί η κλάση SampleWriter του JSyn, ένα φιλτράρισμα αρχείων ώστε να βεβαιωθεί ο αλγόριθμος για τον σωστό τύπο αρχείου ώστε να συνεχίσει να λειτουργεί. Για τον λόγω αυτό δημιουργήθηκε μια κλάση που ονομάστηκε SoundFilesFilter. Αυτή η κλάση με την σειρά της χρησιμοποιεί μία άλλη κλάση για τον ορισμό των αρχείων που χρειαζόμαστε, την κλάση Utils.

Αρχικά φτίαχνουμε μια νέα εκδοχή της κλάσης JFileChooser:

```
fc=new JFileChooser();
```

Αφού πρώτα βέβαια αρχικοποιήσουμε:

```
JFileChooser      fc;
```

Η εμφάνιση αυτού του καινούργιου παραθύρου fc θέλουμε να πυροδοτείται από το πάτημα ενός κουμπιού. Φτιάχνουμε λοιπόν ένα καινούργιο JButton, το ονομάζουμε Open και του προσθέτουμε έναν ActionListener όπως περιγράφεται στην προηγούμενη παράγραφο. Έπειτα προσθέτουμε το φίλτρο αρχείων που φτιάξαμε στην κλάση SoundFilesFilter. Αυτό γίνεται φτιάχνοντας ένα αντικείμενο της κλάσης και δηλώνοντας το στη μέθοδο addChoosableFileFilter. Επιπλέον κάνουμε κάποιες δηλώσεις που εξυπηρετούν στην χρήση του παραθύρου και της πλοήγησης.

```
fc.setDragEnabled(true);
fc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
fc.addChoosableFileFilter(new SoundFilesFilter());
```

Για να δεχθεί η μέθοδος addChoosableFileFilter το αντικείμενο της κλάσης μας SoundFilesFilter πρέπει να δηλώσουμε κατά την δήλωση της κλάσης ότι είναι

επέκταση της κλάσης FileFilter της Java.

```
public class SoundFilesFilter extends FileFilter {…}
```

Το κάθε αντικείμενο της κλάσης SoundFilesFilter επιστρέφει την Boolean τιμή true ή False για κάθε αρχείο που φαίνεται στον JFileChooser για το αν δέχεται ή όχι το συγκεκριμένο αρχείο. Αυτό προκαλεί στον JFileChooser μια διαφορετική σκίαση στα αρχεία που δε δέχεται το φίλτρο και ο χρήστης δεν μπορεί να τα επιλέξει.

```
public boolean accept(File f) {
        if (f.isDirectory()) {
            return true;
        }
        String extension = Utils.getExtension(f);
        if (extension != null) {
            if (extension.equals(Utils.aiff) ||
                extension.equals(Utils.aif) ||
                extension.equals(Utils.wav)){
                    return true;
            } else {
                return false;
            }
        }
        return false;
    }
```

Αυτό το φιλτράρισμα γίνεται με τον έλεγχο της προέκτασης του κάθε αρχείου κάτι που ελέγχεται στην κλάση Utils.

```
public class Utils {
    public final static String wav = "wav";
    public final static String aiff = "aiff";
    public final static String aif = "aif";
    /*
     * Get the extension of a file.
     */
    public static String getExtension(File f) {
        String ext = null;
        String s = f.getName();
        int i = s.lastIndexOf('.');

        if (i > 0 &&  i < s.length() - 1) {
            ext = s.substring(i+1).toLowerCase();
        }
        return ext;
    }
 }
```

Αφού βεβαιωθήκαμε για την σωστή επιλογή του αρχείου πρέπει να επιστρέψουμε αυτο το αρχείο στο SynthSample ώστε να δημιουργηθεί το stream.

```
if (e.getSource() == openButton) {
```

```
int returnVal = fc.showOpenDialog(Enffected.this);
if (returnVal == JFileChooser.APPROVE_OPTION) {
try{
      File file = fc.getSelectedFile();
      stream = (InputStream) (new FileInputStream(file));
      openedFileName=file.getName();

      switch( SynthSample.getFileType(openedFileName) ){
                  case SynthSample.AIFF:
                        mySamp = new SynthSampleAIFF();
                  break;
                  case SynthSample.WAV:
                        mySamp = new SynthSampleWAV();
                  break;
                  default:
                        SynthAlert.showError(this, "Unrecognized sample
filetype. Please open only 16bit mono .aiff or .wav sound files");
                  break;
}
```

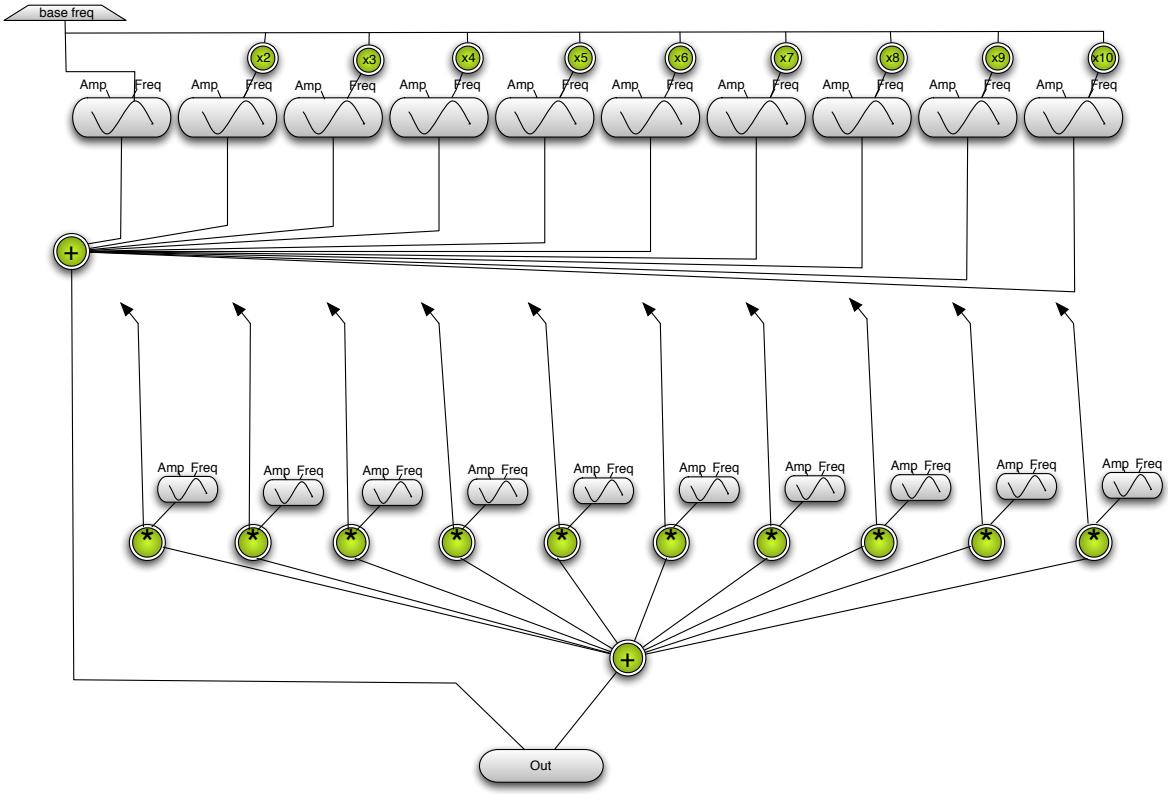
Σημειώνεται ότι κάθε δεκτός τύπου αρχείου (aiff ή wav) πρέπει να φορτωθεί σε διαφορετικό SynthSample καθώς έχουν διαφορετικό file structure και το καθένα ειναι προγραμματισμένο να επεξεργάζεται συγκεκριμένο file structure.

2.2 Η εφαρμογή AmAdd

2.2.1 Περιγραφή της Εφαρμογής

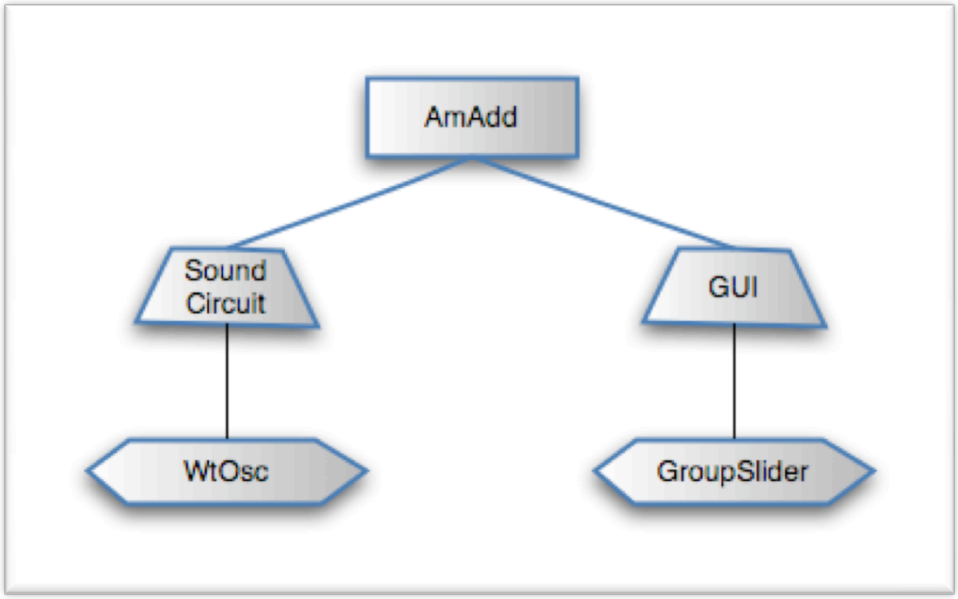
Στην εφαρμογή AmAdd χρησιμοποιούνται τρεις τεχνικές σύνθεσης ήχου. Η wavetable, η προσθετική και η τεχνική διαμόρφωσης πλάτους. Αρχικά γεμίζουμε τον πίνακα με τιμές ώστε να δημιουργήσουμε μια ημιτονοειδή κυματομορφή. Δημιουργώντας δέκα αντικείμενα της κλάσης του πίνακα και ταλαντώνοντας τα με τα ακέραια πολλαπλάσια μιας βασικής συχνότητας, προσθέτοντας τα έχουμε μία κλασική προσθετική τεχνική σύνθεσης ήχου. Υπάρχει και μια επιπλέον επιλογή όπου διαμορφώνεται κατά πλάτος ο κάθε αρμονικός της βασικής συχνότητας.

Ο χρήστης μπορεί να ελέγχει την βασική συχνότητα στην προσθετική σύνθεση, τα πλάτη των αρμονικών, και την συχνότητα της διαμόρφωσης του κάθε αρμονικού. Επίσης επιλέγει την χρήση ή όχι της διαμόρφωσης πλάτους. Επίσης μπορεί να επιλέγει και την κυματομορφή του ταλαντωτή διαμόρφωσης.



Για την υλοποίηση της εφαρμογής χρειάζεται η δημιουργία τριών κλάσεων. Έκτος από την κεντρική κλάση όπου γίνονται οι βασικές συνδέσεις και το κυρίως γραφικό περιβάλλον, δημιουργούμε μια κλάση για τον πίνακα κυματομορφής(WtOsc), και τροποποιούμε μία κλάση που χρησιμοποιήσαμε στην

εφαρμογή despect για ομαδοποιημένα sliders(GroupSlider).



2.2.2 Η Υλοποίηση του ηχητικού κυκλώματος

Αρχικά πρέπει να φτιάξουμε μια κλάση η οποία θα είναι το κύκλωμα του βασικού ταλαντωτή που χρησιμοποιείται στο πρώτο επίπεδο της εφαρμογής δηλαδή στην προσθετική σύνθεση. Η κλάση λοιπόν WtOsc θα πρέπει να είναι ένα ανεξάρτητο κύκλωμα το οποίο θα παρέχει τον έλεγχο της συχνότητας και του πλάτους της τελικής κυματομορφής. Έτσι πρέπει να δηλώσουμε κατά την δήλωση της κλάσης ότι αυτή επεκτείνεται στην κλάση SynthCircuit.

```
public class WtOsc extends SynthCircuit{…}
```

Έπειτα πρέπει να φτιάξουμε τον πίνακα όπου θα γεμίσουμε τιμές ώστε να σχηματίσουμε την ημιτονοειδή κυματομορφή. Το Jsyn παρέχει δύο πολύ χρήσιμες κλάσεις για την συγκεκριμένη τεχνική, την SynthTable ένας πίνακας έτοιμος να δεχθεί τιμές για τον σχηματισμό κυματομορφών και την TableOscillator ένας ταλαντωτής των παραπάνω πινάκων. Φτιάχνουμε λοιπόν τα αντικείμενα τους.

```
myTable = new SynthTable(WAVE_LENGTH + 1);
add(Osc = new TableOscillator());
```

Πρέπει να σημειωθεί ότι σε ένα ηχητικό κύκλωμα μπορούμε να προσθέσουμε (add) μόνο ηχητικές μονάδες όπως το TableOscillator. Το SynthTable το οποίο είναι ένας πίνακας δεν χρειάζεται να γίνει add, απλά χρειάζεται να οριστεί ένα νέο αντικείμενό του. Όταν γίνεται add ένα αντικείμενο σε μια υπό-κλάση SynthCircuit, ξεκινάει η ηχητική επεξεργασία και αυτού, κατά την κλήση Synth.*startEngine*(0) στην κεντρική κλάση.

Πρέπει επίσης να βάλουμε και τα ports για να γίνεται ο έλεγχος του ταλαντωτή, και φυσικά να έχει μια έξοδο σήματος.

```
addPort(frequency = Osc.frequency,"frequency");
addPort(amplitude = Osc.amplitude,"amplitude");
addPort(output = Osc.output,"output");
```

Τέλος πρέπει να ορίσουμε και τον πίνακα που θα χρησιμοποιήσει η SynthTable και να τον γεμίσουμε τιμές ώστε να μας δώσει αποτέλεσμα ένα ημίτονο. Αυτό θα το κάνουμε εφαρμόζοντας την παρακάτω συνάρτηση στον κώδικα.

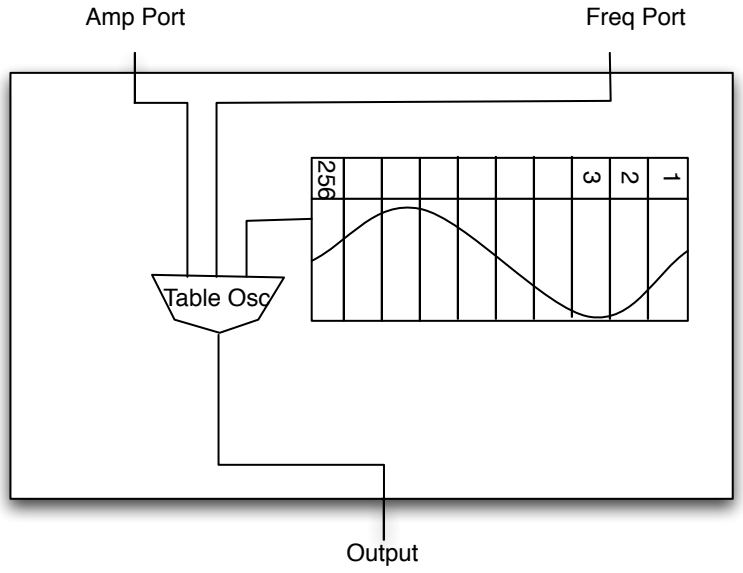
$$y = \sum_{i=0}^{i=WtL} \left(\frac{\sin(\frac{i \times 2\pi}{t})}{2} + \frac{\sin(3 \times \frac{i \times 2\pi}{t})}{2}\right) \times 32767$$

Πρέπει να ορίσουμε και την τελευταία θέση του πίνακα ως θέση ασφαλείας με την τιμή 0

```
data = new short[WAVE_LENGTH+1];
        for( int i=0; i<WAVE_LENGTH; i++ ) {
                                data[i] = (short) (32767.0 *
                (0.5*Math.sin(i*2.0*Math.PI/WAVE_LENGTH)
                                +
                0.5*Math.sin(3.0*i*2.0*Math.PI/WAVE_LENGTH)));
        }

    data[WAVE_LENGTH] = data[0];
        myTable.write( data );
```

Τέλος θέτουμε το SynthTable από το οποίο θα διαβάσει ο TableOscillator την κυματομορφή.

```
        Osc.tablePort.setTable( myTable );
```



Έχοντας φτιάξει τον βασικό ταλαντωτή της εφαρμογής μπορούμε να κάνουμε κάποιους συνδυασμούς στην κεντρική κλάση ώστε να φτιάξουμε την τελική μηχανή σύνθεσης ήχου. Δημιουργούμε λοιπόν 10 αντικείμενα της κλάσης που μόλις φτιάξαμε για να γίνει η πρόσθεση τους σε ένα bus. Μετά τους 10 ταλαντωτές που φτιάξαμε, και πριν από τις εισόδους του bus writer παρεμβάλουμε και 10 αντικείμενα ενός ταλαντωτή του Jsyn, του SineOscillator. Οι SineOscillators θα μας δίνουν την δυνατότητα να διαμορφώσουμε τους αρχικούς ταλαντωτές ξεχωριστά. Εκτός από τους SineOscillators δημιουργούμε και ταλαντωτές με διαφορετικές κυματομορφές ώστε να έχουμε την επιλογή αργότερα να αλλάζουμε το τελικό ηχόχρωμα της διαμόρφωσης.

```
myBusReader = new BusReader();
OscC = new WtOsc[NUM_OSCS];
OscM = new SineOscillator[NUM_OSCS];
OscM1 = new SawtoothOscillator[NUM_OSCS];
OscM2 = new SquareOscillatorBL[NUM_OSCS];
OscM3 = new TriangleOscillator[NUM_OSCS];
out = new LineOut();
addu= new AddUnit[NUM_OSCS];
myBusWriter= new BusWriter[NUM_OSCS];
mfriv = new double[NUM_OSCS];
for(int i=0;i<10;i++){
      OscC[i] = new WtOsc();
      OscM[i] = new SineOscillator();
      OscM1[i] = new SawtoothOscillator();
      OscM2[i] = new SquareOscillatorBL();
      OscM3[i] = new TriangleOscillator();
      addu[i] = new AddUnit();
      OscC[i].frequency.set(freq*i+freq);
      OscC[i].amplitude.set(0);
      OscM[i].output.connect(addu[i].inputB);
      addu[i].output.connect(OscC[i].amplitude);
      OscM[i].amplitude.set(0);
      myBusWriter[i] = new BusWriter();
      OscC[i].output.connect( myBusWriter[i].input );
myBusWriter[i].busOutput.connect(myBusReader.busInput );
}
```

2.2.3 Η Υλοποίηση του Graphical User Interface

Για την καλύτερη χρηστικότητα της εφαρμογής αποφασίστηκε ο έλεγχος λίγων βασικών παραμέτρων που έχουν δραστική επίδραση στον τελικό ήχο της εφαρμογής. Αυτές οι παράμετροι είναι η βασική συχνότητα της προσθετικής σύνθεσης, τα πλάτη των αρμονικών, οι συχνότητες της διαμόρφωσης των αρμονικών, η επιλογή των κυματομορφών διαμόρφωσης και δυο επιλογές: να αρχίζει και να σταματάει η λειτουργία της εφαρμογής και της διαμόρφωσης.

Για τα πλάτη των αρμονικών και τις συχνότητες της διαμόρφωσης τους θα τα ελέγχουμε μετατρέποντας μια κλάση που ήδη έχουμε φτιάξει για την εφαρμογή

despect, την κλάση GroupSlider. Η μετατροπή αφορά στα κουμπιά rand τα οποία αυτή την φορά δεν χρειάζονται, καθώς κάθε μια μετακίνηση slider θα δίνει συγκεκριμένη και πολύ κατανοητή αλλαγή στο τελικό σήμα. Επίσης αυτήν την φορά έχουμε μικρότερο όγκο παραμέτρων που ελέγχουμε και γίνεται γενικότερα πιο εύκολη η χρήση της εφαρμογής.

Για την επιλογή των κυματομορφών χρησιμοποιούμε την κλάση JComboBox του Swing. Το JComboBox είναι μια λίστα η οποία δίνει ένα action Event για κάθε επιλογή που γίνεται από τον χρήστη, και διαχωρίζει την κάθε επιλογή αναλόγως με το όνομα που έχουμε δώσει. Πρέπει πρώτα να δώσουμε έναν πίνακα με strings για να ονομάσουμε τις επιλογές.

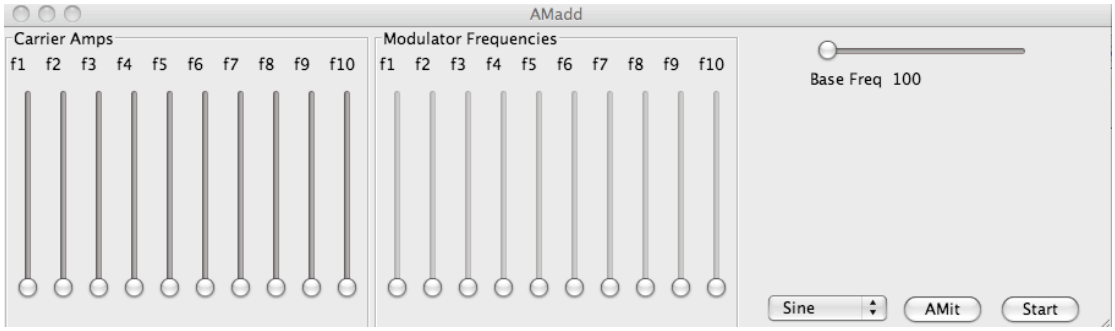
```
String[] wforms = {"Sine", "Sawtooth", "Square", "Triangle"};

cbx = new JComboBox(wforms);
cbx.addActionListener(this);
```

Έτσι μπορούμε στην μέθοδο ActionPrformed να επιλέξουμε κάθε φορά την λειτουργία που θέλουμε αναλόγως το όνομα που έχουμε δώσει στην λίστα του JComboBox.

```
public void actionPerformed(ActionEvent e) {
 if (e.getSource() == cbx){
 JComboBox cb = (JComboBox)e.getSource();
      String wName = (String)cb.getSelectedItem();
      if (wName == "Sawtooth"){
            for(int i=0;i<10;i++){
                  OscM[i].output.disconnect();
                  OscM2[i].output.disconnect();
                  OscM3[i].output.disconnect();
                  OscM1[i].output.connect(addu[i].inputB);
            }
      }
      if (wName == "Square"){
            for(int i=0;i<10;i++){
                  OscM[i].output.disconnect();
                  OscM1[i].output.disconnect();
                  OscM3[i].output.disconnect();
                  OscM2[i].output.connect(addu[i].inputB);
            }
      }…
```

Εδώ επιλέγοντας μια νέα κυματομορφή για τον ταλαντωτή διαμόρφωσης αποσυνδέουμε οποιονδήποτε άλλο ταλαντωτή που μπορεί να ήταν συνδεδεμένος πριν και μετά συνδέουμε αυτόν που θέλουμε.

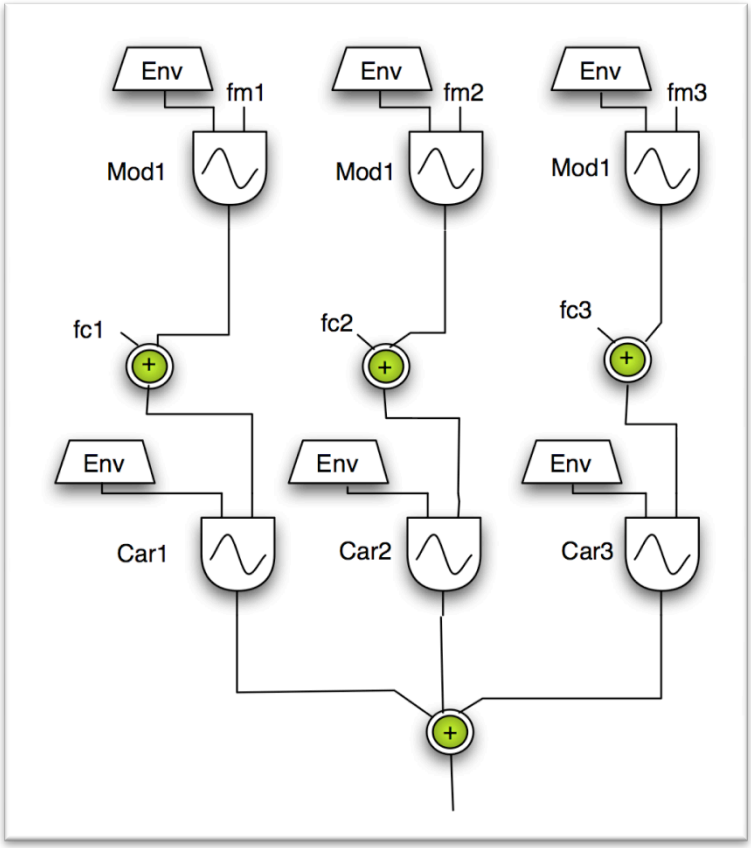
Τοποθετούμε λοιπόν όλα τα αντικείμενα σε στο κεντρικό JPanel και αυτό με την σειρά του στο JFrame της εφαρμογής AmAdd.


AMadd
Carrier Amps
f1 f2 f3 f4 f5 f6 f7 f8 f9 f10
Modulator Frequencies
f1 f2 f3 f4 f5 f6 f7 f8 f9 f10
Base Freq 100
Sine
AMit
Start

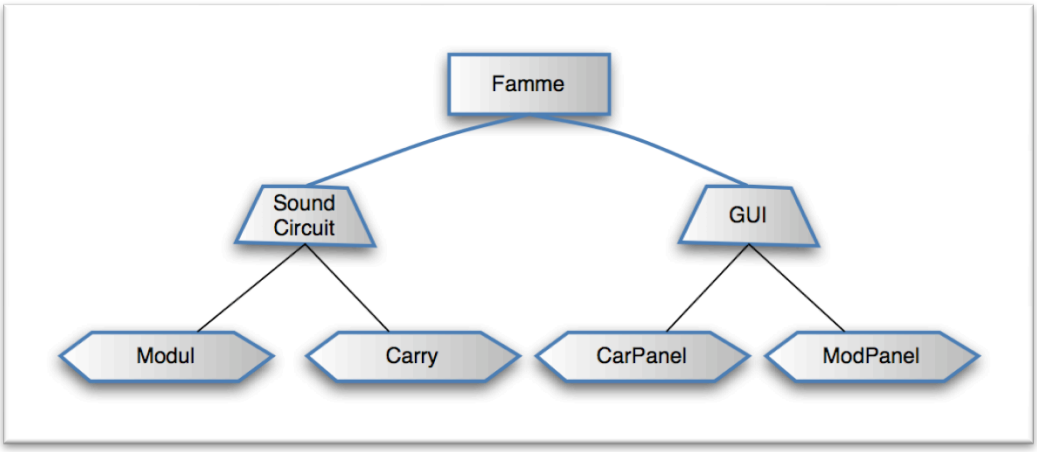
2.3 Η εφαρμογή Famme

2.3.1 Περιγραφή της Εφαρμογής

Η τρίτη εφαρμογή έχει σχεδιαστεί εξ' ολοκλήρου με την τεχνική διαμόρφωσης συχνότητας(FM) η οποία περιλαμβάνει τόσο την κλασική FM όσο και την MCFM(. Πρόκειται για τρία ζεύγη ταλαντωτών όπου σε κάθε ζεύγος ο ένας ταλαντώνει την συχνότητα του άλλου ή διαφορετικά τα ζεύγη αποτελούνται από δύο είδη ταλαντωτών, τους διαμορφωτές(Modulators) και τους φέροντες(Carriers). Στην περίπτωση της FM σύνθεσης το JSyn παρέχει μια εξειδικευμένη κλάση η οποία ονομάζεται FMOperator. Ο FMOperator μπορεί να χρησιμοποιηθεί και για τις δυο περιπτώσεις ταλαντωτή καθώς είναι σχεδιασμένος έτσι ώστε να έχει είσοδο σήματος, ρυθμίσεις για το πλάτος και την συχνότητα για τον carrier και ρυθμίσεις για το βάθος ταλάντωσης(d) και την συχνότητα διαμόρφωσης για τον modulator. Ακόμα μπορεί η έξοδος του να συνδεθεί κατευθείαν με την είσοδο του ώστε να υπάρχει η δυνατότητα της ανατροφοδότησης ενός modulator.



Η εφαρμογή αποτελείται από πέντε κλάσεις συνολικά. Η κεντρική κλάση Famme περιέχει όλες τις βασικές συνδέσεις τών μονάδων και τις μεθόδους διάδρασης του γραφικού περιβάλλοντος. Οι δύο κλάσεις Modul και Carry είναι ηχητικές υπομονάδες που περιέχουν τους FMOperators. Τέλος οι κλάσεις CarPanel και ModPanel είναι τα γραφικά περιβάλλοντα του κάθε ταλαντωτή. Περιέχουν δύο sliders για ρυθμίσεις , μία περιβάλλουσα η οποία εμφανίζεται με το κουμπί "Show Envelopes" της κεντρικής κλάσης και τέλος ένα κουμπί για το σβήσιμο των σημείων στο γραφικό της περιβάλλουσας



.

2.3.2 Η Υλοποίηση του ηχητικού κυκλώματος

Αρχίζοντας από τον ταλαντωτή διαμόρφωσης φτίαχνουμε μια κλάση που επεκτείνεται στην κλάση SynthCircuit και την ονομάζουμε Modul. Αυτή θέλουμε να περιέχει τον FMOperator και την περιβάλλουσα για το index. Η περιβάλλουσα παρέχεται απο την κλάση SynthEnvelope και για να χρησιμοποιηθεί χρειάζεται κάποιες άλλες κλάσεις όπως την EnvelopePlayer, την EnvelopeEditor και την EnvelopePoints. Η πρώτη είναι αυτή που αναπαράγει τον πίνακα της SynthEnvelope, η δεύτερη είναι ένας γραφικός πίνακας όπου μπορεί κάποιος να σχεδιάσει με το ποντίκι κάποια περιβάλλουσα και τέλος η τρίτη είναι τα σημεία τα οποία αποθηκεύονται στον πίνακα της SynthEnvelope μέσω του EnvelopeEditor. Για την λειτουργία του EnvelopeEditor χρειάζεται η κληρονόμηση της κλάσης EditListener, οπότε έχουμε:

```
public class Modul extends SynthCircuit implements EditListener{…
```

Δημιουργούμε τώρα τα αντικείμενα FMOperator, τα αντικείμενα για την περιβάλλουσα και όποια χρειάζεται να εκκινηθούν, συμπληρώνουμε την έκφραση add στην αρχή ώστε να εκκινηθούν απο την κλήση start() του αντικειμένου της κλάσης Modul στην κεντρική κλάση. Σημειώνεται ότι ο FMOperator έχει ένα πεδίο όπου δηλώνοντας τον SynthEnvelope που δηλώνουμε, αναλαμβάνει ο ίδιος να το αναπαράξει. Έτσι η χρήση του EnvelopePlayer δεν είναι απαραίτητη.

```
add(osc = new FMOperator());
envelope = new SynthEnvelope( MAX_FRAMES );
envedit = new EnvelopeEditor();
envedit.setBackground(Color.black);
envedit.setForeground(Color.green.brighter());
envedit.setVerticalBarsEnabled(true);
envedit.setVerticalBarSpacing(0.1);
points = new EnvelopePoints();
points.add( 0.1, 1.0 );
points.add( 0.2, 0.6 );
points.add( 0.5, 0.55 );
points.add( 0.3, 0.0 );
```

```
            envedit.setPoints(points);
            osc.setEnvelope(envelope);
            addPort(output = osc.output, "output" );
```

Πρέπει να γράψουμε και κάποιες μεθόδους που αφορούνε στην συμπεριφορά της περιβάλλουσας.

```
      public void objectEdited( Object editor, Object objPoints ){
            if( looping ){
                  doQueueLoop();
            }
      }
      public void updateEnvelope(){
            int numFrames = points.size();
            for( int i=0; i<numFrames; i++ ){
                  envelope.write( i, points.getPoint( i ), 0, 1 );
            }
      }
      public void doQueue(){
            updateEnvelope();
            osc.envelopePort.queue( envelope, 0, points.size() );
            looping = false;
      }
      public void doQueueLoop(){
            updateEnvelope();
            osc.envelopePort.queueLoop( envelope, 0, points.size() );
            looping = true;
      }
      public void doClear(){
            osc.envelopePort.clear();
            looping = false;
      }
}
```

Η μέθοδος doClear αφαιρεί το Port του envelope κάτι που σημαίνει ότι με την ενεργοποίηση του το σήμα στην έξοδο μας θα παραμείνει σταθερό μέχρι την επόμενη ενεργοποίηση του port, κάτι που γίνεται στις μεθόδους doQueue και doQueueLoop. Η πρώτη απο αυτές αναπαράγει την περιβάλλουσα μια φορά ενώ η επόμενη το κάνει επαναλαμβανόμενα. Η μέθοδος updateEnvelope απλά επαναφέρει τα σημεία του SynthEnvelope στην αρχηκή τους θέση, και τέλος η objectEdited καλεί την μέθοδο doQueueLoop.

Η επόμενη κλάση που φτιάχνουμε είναι η Carry. Αυτή επίσης θα περιέχει τον FMOperator και την περιβάλλουσα και έναν πολαπλάσιαστή σήματος ο οποίος θα είναι υπεύθυνος για την μη παραμόρφωση του τελικού σήματος στη έξοδο.

```
public class Carry extends SynthCircuit implements EditListener{
...
```

```
Carry(){
	add(cosc = new FMOperator());
	add(utf = new MultiplyUnit());
	envelope = new SynthEnvelope( MAX_FRAMES );
	envedit = new EnvelopeEditor();
	envedit.setBackground(Color.black);
	envedit.setForeground(Color.green.brighter());
	envedit.setVerticalBarsEnabled(true);
	envedit.setVerticalBarSpacing(0.1);
	points = new EnvelopePoints();
	points.add( 0.1, 1.0 );
	points.add( 0.2, 0.6 );
	points.add( 0.5, 0.55 );
	points.add( 0.3, 0.0 );
	envedit.setPoints(points);
	cosc.setEnvelope(envelope);
	cosc.output.connect(utf.inputA);
	utf.inputB.set(0.3);
	addPort(input = cosc.input,"input");
	addPort(output = utf.output,"output");

}
```

Στην κεντρική κλάση τώρα θα γίνουν όλες οι συνδέσεις. Επίσης βάζουμε κάποιους διαύλους για την ομαλή λειτουργία των συνδέσεων. Στην ουσία βάζουμε έναν δίαυλο πρίν απο κάθε carrier για να είναι έτοιμοι να δεχθούν κάποιο άλλο σήμα σε περίπτωση που το χρειαστούμε.

```
for(int i=0; i<3; i++){
	modul[0].output.connect(abus1[i].input);
	modul[1].output.connect(abus2[i].input);
	modul[2].output.connect(abus3[i].input);
	carry[i].output.connect(abus[i].input);
	carry[i].cosc.depth.set(0);
	modul[i].osc.frequency.set(40);
	carry[i].cosc.frequency.set(400);
	carry[i].cosc.amplitude.set(0);
	isParallel[i]=true;
	mated[i] = false;
	isSer[i] = false;
	abus[i].busOutput.connect(buso.busInput);
	abus1[i].busOutput.connect(buso1.busInput);
	abus2[i].busOutput.connect(buso2.busInput);
	abus3[i].busOutput.connect(buso3.busInput);
	buso1.output.connect(carry[0].input);
	buso2.output.connect(carry[1].input);
	buso3.output.connect(carry[2].input);
	buso.output.connect(0,lineout.input,0);
	buso.output.connect(0,lineout.input,1);

	abus[i].start();
	abus1[i].start();
```

```
                abus2[i].start();
                abus3[i].start();
                buso.start();
                buso1.start();
                buso2.start();
                buso3.start();
                modul[i].start();
                carry[i].start();
                lineout.start();

        }
```

2.2.3 Η Υλοποίηση του Graphical User Interface

Το γραφικό περιβάλλον που θα έρχεται σε επαφή ο χρήστης θα έχει ένα πάνελ για κάθε έναν ταλαντωτή, το οποίο θα περιέχει από δύο sliders για τον έλεγχο των παραμέτρων, έναν γραφικό πίνακα για την περιβάλλουσα και ένα κουμπί για τον καθαρισμό των σημείων του πίνακα αυτού. Επιπλέον θα έχει και κουμπιά γενικού ελεγχου για την εμφάνιση των γραφικών πινάκων, το παίξιμο μιας νότας, και το πάγωμα της νότας αυτής.

Αρχικά φτιάχνουμε μια κλάση για το πάνελ των ταλαντωτών διαμόρφωσης και μια άλλη για το πάνελ των φερόντων.

```
public class ModPanel extends JPanel {
```

Στον constructor δηλώνουμε τις παραμέτρους οι οποίες θέλουμε να διαμορφώνουν την κάθε εκδοχή αντικείμένου της κλάσης αυτής. Οι παράμετροι αυτές θα είναι η μέγιστη και η ελάχιστη τιμή των δυό sliders. Τοποθετούμε επίσης όλα τα απαιτούμενα γραφικά στοιχεία που χρειαζόμαστε και όπου χρειάζεται δηλώνουμε Layout Manager που να βολεύει για την καλή τοποθέτηση των στοιχείων αυτών στο panel.

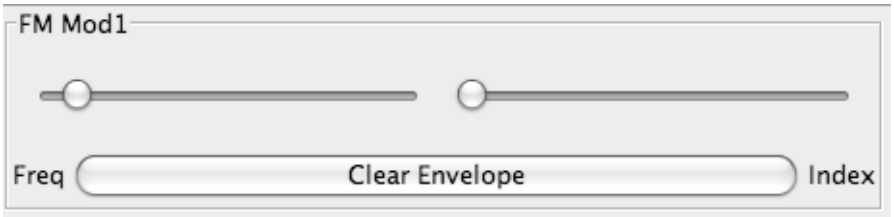
```
public ModPanel(int modnum, int freqmin,
              int freqmax , int indexmin , int indexmax){
            mnum = modnum;
            this.setLayout(new BorderLayout());
            this.setBorder(new TitledBorder(new EtchedBorder(),"FM
            Mod" + mnum));
            JPanel paneldown = new JPanel(new BorderLayout());
            JPanel panelmain = new JPanel(new GridLayout(0,2));
            JPanel panelup = new JPanel(new GridLayout(0,3));
            fknob = new JSlider(freqmin,freqmax,40);
            iknob = new JSlider(indexmin,indexmax,0);

            flabel = new JLabel(" ");
            ilabel = new JLabel(" ");
            panelmain.add(fknob);
            panelmain.add(iknob);
            paneldown.add(flabel, BorderLayout.LINE_START);
            paneldown.add(clear,BorderLayout.CENTER);
            paneldown.add(ilabel,BorderLayout.LINE_END);
            this.add(panelmain,BorderLayout.CENTER);
```

```
this.add(paneldown,BorderLayout.SOUTH);
this.add(panelup,BorderLayout.NORTH);
```

Ομοίως και στην κλάση για το άλλο πάνελ:

```
public CarPanel(int modnum, int freqmin,
                    int freqmax , int ampmin , int ampmax){
        mnum = modnum;
        this.setLayout(new BorderLayout());
        this.setBorder(new TitledBorder(new EtchedBorder(),"FM
        Car" + mnum));
        JPanel paneldown = new JPanel(new BorderLayout());
        JPanel panelmain = new JPanel(new GridLayout(0,2));
        fknob = new JSlider(freqmin,freqmax,400);
        aknob = new JSlider(ampmin,ampmax,0);
        flabel = new JLabel(" ");
        alabel = new JLabel(" ");
        clear = new JButton("Clear Envelope");
        panelmain.add(fknob);
        panelmain.add(aknob);
        paneldown.add(flabel,BorderLayout.WEST);
        paneldown.add(alabel,BorderLayout.EAST);
        paneldown.add(clear);
        this.add(panelmain,BorderLayout.CENTER);
        this.add(paneldown,BorderLayout.SOUTH);
}
```

Τα πάνελ θα έχουν την παρακάτω μορφή.



Τώρα στην κεντρική κλάση θα εισάγουμε όλα τα στοιχεία του κεντρικού παραθύρου της εφαρμογής, θα φτιάξουμε την λειτουργία της εμφάνισης των γραφικών πινάκων των envelopes και τέλος θα κάνουμε τα handle των Listeners.

Προσθέτουμε λοιπόν στο παράθυρο όλα τα αντικείμενα των κλάσεων που φτιάξαμε, και τα υπόλοιπα που θα χρειαστούμε.

```
toE = new JToggleButton("Show Envelopes");
toE.addActionListener(this);
queButton = new JButton ("PlayNote");
freeze = new JButton ("Freeze");
freeze.addActionListener(this);
queButton.addActionListener(this);

      lowerFrame.add(toE);
```

```
lowerFrame.add(queButton);
lowerFrame.add(freeze);
this.add(initFrame, BorderLayout.CENTER);
this.add(lowerFrame, BorderLayout.SOUTH);
//c.output.connect(d.inputB);
ep11 = new JPanel();
ep12 = new JPanel();
ep11.setBorder(new TitledBorder(new
EtchedBorder(),"Index"));
ep12.setBorder(new TitledBorder(new
EtchedBorder(),"Amp"));
ep11.setLayout(new GridLayout(1,0));
ep12.setLayout(new GridLayout(1,0));
ep11.add(modul[0].envedit);
ep12.add(carry[0].envedit);
envFrame1.add(ep11);
envFrame1.add(ep12);
ep21 = new JPanel();
ep22 = new JPanel();
ep21.setBorder(new TitledBorder(new
EtchedBorder(),"Index"));
ep22.setBorder(new TitledBorder(new
EtchedBorder(),"Amp"));
ep21.setLayout(new GridLayout(1,0));
ep22.setLayout(new GridLayout(1,0));
ep21.add(modul[1].envedit);
ep22.add(carry[1].envedit);
envFrame1.add(ep21);
envFrame1.add(ep22);
ep31 = new JPanel();
ep32 = new JPanel();
ep31.setBorder(new TitledBorder(new
EtchedBorder(),"Index"));
ep32.setBorder(new TitledBorder(new
EtchedBorder(),"Amp"));
ep31.setLayout(new GridLayout(1,0));
ep32.setLayout(new GridLayout(1,0));
ep31.add(modul[2].envedit);
ep32.add(carry[2].envedit);
envFrame1.add(ep31);
envFrame1.add(ep32);
```

Τα πάνελ ep τα βάζουμε ετσι ώστε να μπορέσουμε να διαχωρίσουμε envelopeEditors μεταξύ τους καθώς έχουμε δικαίωμα στα Jpanels να βάλουμε σύνορα (ep32.setBorder) ενώ σταν awt envelopeEditors όχι.

Φτιάχνουμε τώρα μια μέθοδο με την οποία θα μπορούμε να αντιστοιχούμε ενέργειες για κάθε κουμπί.

```
public void actionPerformed(ActionEvent e) {

	for(int i = 0; i < 3; i++){
		if (e.getSource() == queButton){
			modul[i].doQueue();
			carry[i].doQueue();
		}
		if (e.getSource() == freeze){
			modul[i].doClear();
			carry[i].doClear();
		}
		if (e.getSource() == mod[i].clear){
			modul[i].points.clear();
		}
		if (e.getSource() == car[i].clear){
			carry[i].points.clear();
		}
		if (e.getSource() == toE){
			JToggleButton source = (JToggleButton)
			e.getSource();
			if(source.isSelected()){
				this.remove(initFrame);
				this.add(envFrame1, BorderLayout.CENTER);
				this.updateUI();
			}else{
				this.remove(envFrame1);
				this.add(initFrame, BorderLayout.CENTER);
				this.updateUI();}
		}
	}
}
```

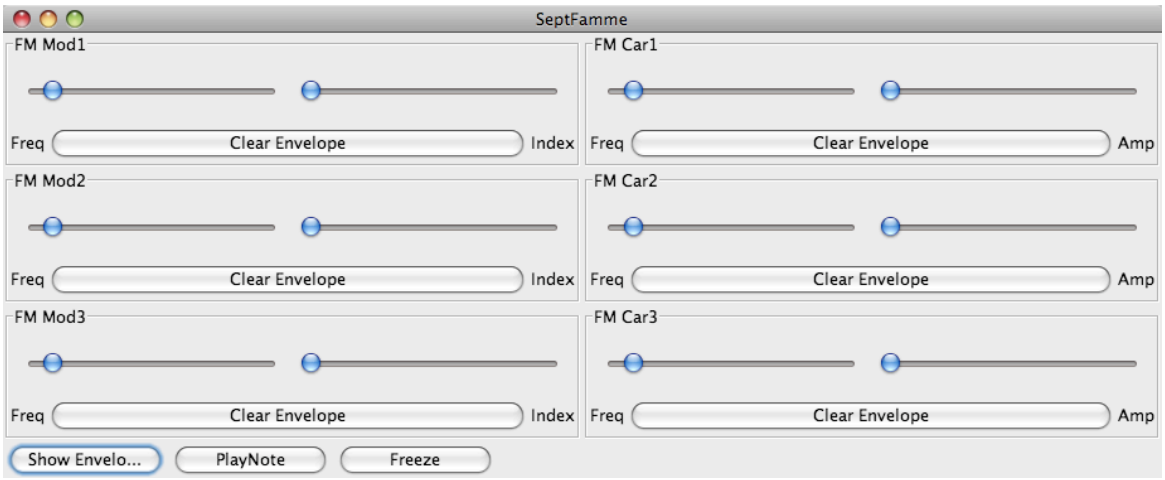
Σημειώνεται πώς το κουμπί toE είναι JToggleButton και η λειτουργία του είναι να παραμένει σε θέση on με ένα πάτημα και με το επόμενο πάτημα να βγαίνει σε θέση off. Την λειτουργία αυτή την χρησιμοποιούμε για να εμφανισουμε και να κρύψουμε τους envelopeEditors.

Τέλος με την μέθοδο stateChanged θα αντιστοιχίσουμε τις κινήσεις των sliders σε ενέργειες πάνω στην μηχανή του ήχου.

```
public void stateChanged(ChangeEvent e) {
		for(int i = 0; i < 3; i++){
			if(e.getSource() == mod[i].fknob){
				JSlider source = (JSlider) e.getSource();
				if (source.getValueIsAdjusting()) {
					fvalue =(double) mod[i].fknob.getValue();

					mod[i].flabel.setText( "Freq" + fvalue);
					modul[i].osc.frequency.set(fvalue);
				}
			}
			if(e.getSource() == mod[i].iknob){
```

```
                JSlider source = (JSlider) e.getSource();
                if (source.getValueIsAdjusting()) {
                        double initvalue =
                        (double)(mod[i].iknob.getValue());
                        ivalue = initvalue/50;
                        mod[i].ilabel.setText( "Index" + ivalue);
                        carry[i].cosc.depth.set(ivalue*fvalue);

                }
        }
        if(e.getSource() == car[i].fknob){
                JSlider source = (JSlider) e.getSource();
                if (source.getValueIsAdjusting()) {
                        double cafvalue =
                        (double)(car[i].fknob.getValue());
                        car[i].flabel.setText( "Freq" + cafvalue);
                        carry[i].cosc.frequency.set(cafvalue);
                }
        }

        if(e.getSource() == car[i].aknob){
                JSlider source = (JSlider) e.getSource();
                if (source.getValueIsAdjusting()) {
                        double initvalue =
                        (double)(car[i].aknob.getValue());
                        //mod.flabel.setVisible(true);
                        cavalue = initvalue/1000;
                        car[i].alabel.setText( "Amp" + cavalue);
                        carry[i].cosc.amplitude.set(cavalue);
                }
        }
}
}
```

Η τελική εμφάνιση της εφαρμογής είναι η εξης:



Συμπεράσματα

Χρησιμοποιώντας την γλώσσα Java και το Jsyn καταλήξαμε σε διάφορα συμπεράσματα σε σχέση με την χρήση του και την ηχητική ποιότητα που προσφέρει.

Το JSyn ως API γλώσσας προγραμματισμού μπορεί να θεωρηθεί δυσκολότερο από άλλα περιβάλλοντα προγραμματισμού (Pd, Max/MSP) για να μάθει κανείς να το χρησιμοποιεί καθώς πρέπει να ξέρει μια γλώσσα προγραμματισμού κάτι που σημαίνει ότι είναι πιο χαμηλού επιπέδου από προγραμματιστικά περιβάλλοντα σύνθεσης ήχου, γραφικά ή μη. Βέβαια αυτό σημαίνει ότι κάποιος έχει περισσότερες επιλογές και μεγαλύτερη ακρίβεια για μία εφαρμογή γιατί μπορεί να χρησιμοποιήσει πλήθος πακέτων και κλάσεων που δύσκολα συναντάται σε προγραμματιστικά περιβάλλοντα. Υπάρχει επίσης ένα πλήρες documentation με πολύ καλά tutorial που βοηθάνε πολύ κάποιον που θέλει να ασχοληθεί. Μια πολύ ενεργή κοινότητα μπορεί να βοηθήσει κάποιον σε οποιδήποτε επίπεδο και αν είναι, οποιαδήποτε στιγμή μέσα από ένα mailing list.

Τα γραφικά αντικείμενα που προσφέρει το JSyn όπως sliders, knobs είναι μάλλον δύσχρηστα και έχουν ένα look and feel για όλα τα συστήματα καθώς έχουν σχεδιαστεί με το παλιό πακέτο γραφικών της Java, awt και υπάρχουν δυσλειτουργίες όταν κάποιος αποφασίσει να τα χρησιμοποιήσει σε συνδυασμό με το πακέτο swing της Java. Για τον λόγο αυτό αποφασίστηκε η χρήση γραφικών μόνο από το πακέτο swing της Java.

Η χρήση των envelopes που είναι σημαντική για κάθε εφαρμογή σύνθεσης ήχου, γίνεται δύσκολα καθώς υπάρχουν τέσσερις διαφορετικές κλάσεις που πρέπει να προστεθούν στον κώδικα ώστε να αναπαραχθεί ένα envelope στην εφαρμογή. Απο την άλλη αυτό σημαίνει πως έχει δοθεί μεγάλη έμφαση στα envelopes και μπορεί να γίνει αρκετά λεπτομερής προγραμματισμός στον τομέα αυτόν.

Ένα ακόμα χαρακτηριστικό του JSyn είναι πως για μερικές τεχνικές μπορεί κάποιος να χρησιμοποιήσει μόνο ορισμένες κλάσεις που παρέχει το πακέτο. Για παράδειγμα στην FM σύνθεση μπορεί κάποιος να χρησιμοποιήσει την κλάση FMOperator και την κλάση FMPair. Η χρήση απλού oscillator δεν ενδύκνειται για μια τέτοια διεργασία. Ακόμα, στις τεχνικές με τις σειρές καθυστέρησης κάποιος μπορεί να μεταβάλλει σε πραγματικό χρόνο τον χρόνο καθυστέρησης μόνο με την χρήση της κλασης InterpolatingDelayUnit. Κάθε προσπάθεια για την αλλαγή της παραμέτρου αυτής με κάποια άλλη κλάση σχετική με τα delay lines θα καταλήξει σε ένα NullPointerException.

Το αρνητικό σημείο του native κώδικα που παράγει το JSyn είναι ότι κατα κάποιο τρόπο εμποδίζει την multi-platform λειτουργία της Java καθώς κάθε εφαρμογή που παράγεται πρέπει να συνοδεύεται με τα κατάλληλα αρχεία διαφορετικά για κάθε πλατφόρμα.

Τέλος, ενώ το πακέτο είναι δωρεάν και ελεύθερο για την χρήση του απο προγραμματιστές, για πολύ συγκεκριμένες λειτουργίες (π.χ. πολυκάναλος ήχος) απαιτείται μια συνδρομή.

Παρ' όλα αυτά προγραμματίζοντας εφαρμογές σύνθεσης ήχου στην Java και το JSyn δίνει την αίσθηση της απόλυτης ελευθερίας σε σχέση με τις προγραμματιστικες δυνατότητες και την αίσθηση της σταθερότητας και μιας σοβαρής μηχανής ήχου σε σχέση με την ποιότητα των εφαρμογών που μπορεί κάποιος να φτιάξει, σε σχέση πάντα με άλλα προγραμματιστικά περιβάλλοντα.

Παράρτημα Α:

Μερικά βασικά στοιχεία για το Java Swing και τους Java Listeners

Κλάσεις, Αντικείμενα, Μεθοδοι

Η γλώσσα προγραμματισμού Java είναι μια αντικειμενοστρεφής γλώσσα, που σημαίνει ότι βασίζεται στην δημιουργία αντικειμένων μιας εφαρμογής για την ανάπτυξή της. Κάθε κλάση που δημιουργείται μπορεί να παράγει αντικείμενα της που έχουν κοινές λειτουργίες αλλά διαφορετικά χαρακτηριστικά. Τα αντικείμενα περιέχουν δεδομένα (μεταβλητές) και κώδικα(μεθόδους). Στην Java σχεδόν κάθε μεταβλητή είναι ένα αντικείμενο κάποιου τύπου, ακόμα και οι συμβολοσειρές (strings).

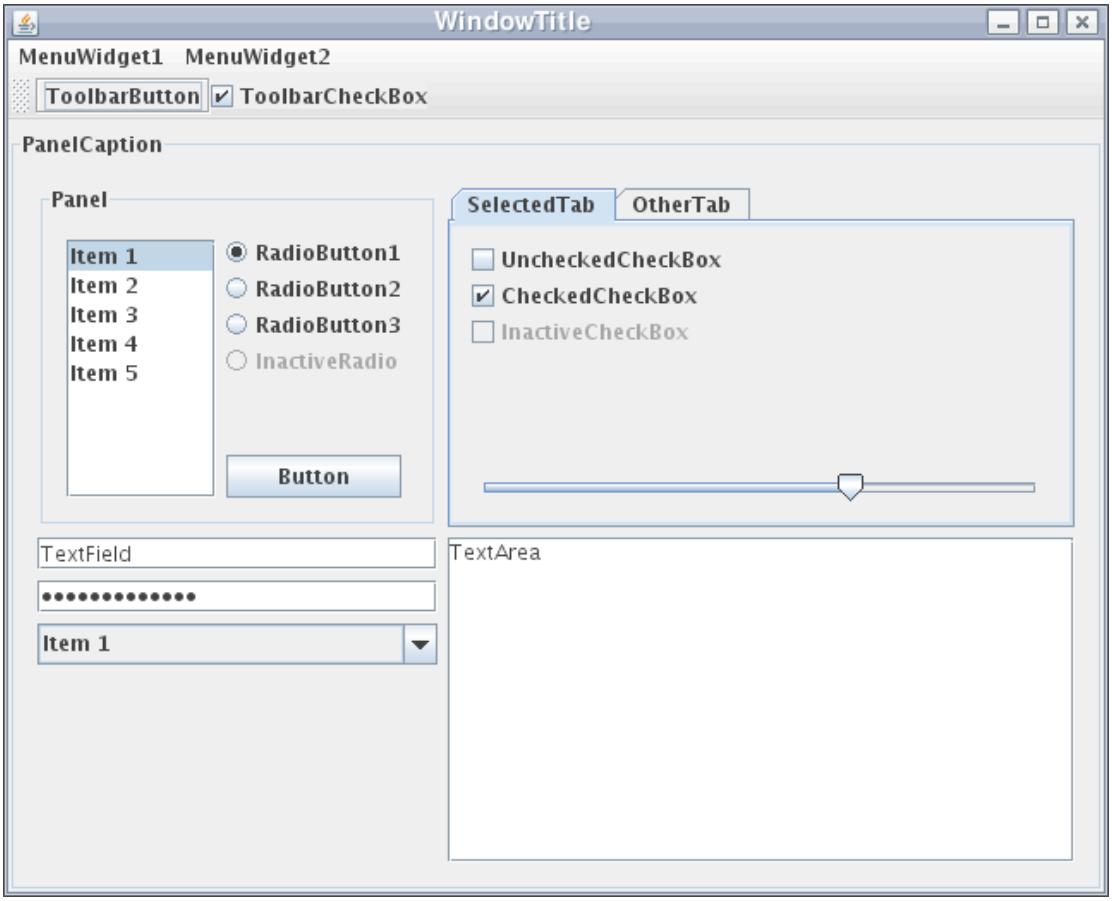
Μια μέθοδος είναι ένα μέρος κώδικα στο οποίο αναφέρεται κάποιος με το συγκεκριμένο όνομα που του έχει δοθεί, και μπορεί να χρησιμοποιηθεί σε οποιοδήποτε σημείο σε ένα πρόγραμμα απλά γράφοντας το όνομα αυτό.
Είναι ένα υποπρόγραμμα το οποίο δρα σε δεδομένα και μπορεί να επιστρέφει τιμές ή απλά να κάνει κάποιες ενέργειες για το πρόγραμμα.

Η Java παρέχει πλήρη πακέτα με κλάσεις αναλόγως την χρήση τους για την διευκόλυνση της ανάπτυξης προγραμμάτων.

Χρήσιμα πακέτα-Swing

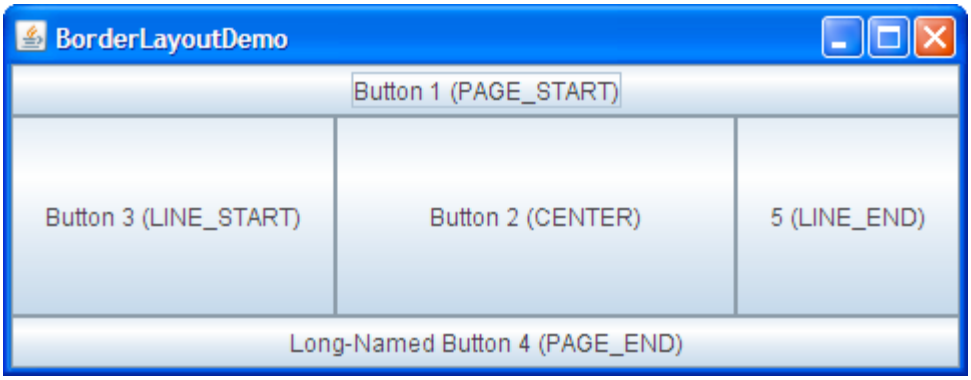
Ένα από τα πιο χρήσιμα πακέτα για την ανάπτυξη προγραμμάτων στην Java είναι το πακέτο Javax.swing. Το πακέτο αυτό αφορά σε όλα τα γραφικά που χρειάζονται για την ανάπτυξη μιας παραθυρικής εφαρμογής ή applet και στην διάδραση της με τον χρήστη. Το πακέτο αυτό είναι η εξέλιξη του πακέτου awt της Java.

Το swing παρέχει το "native look and feel" που σημαίνει ότι τα γραφικά των εφαρμογών θα μοιάζουν με εκείνα των εφαρμογών του συστήματος στο οποίο τρέχει εκείνη τη στιγμή το πρόγραμμα. Επίσης παρέχει και άλλα πρότυπα γραφικών look and feel αναλόγως τις ανάγκες του προγραμματιστή. Το παρακάτω σχήμα δείχνει ένα παράδειγμα γραφικά απεικονιζόμενης εφαρμογής και δείχνει πολλά αντικείμενα από τις κλάσεις του swing που μπορούν να χρησιμοποιηθούν.

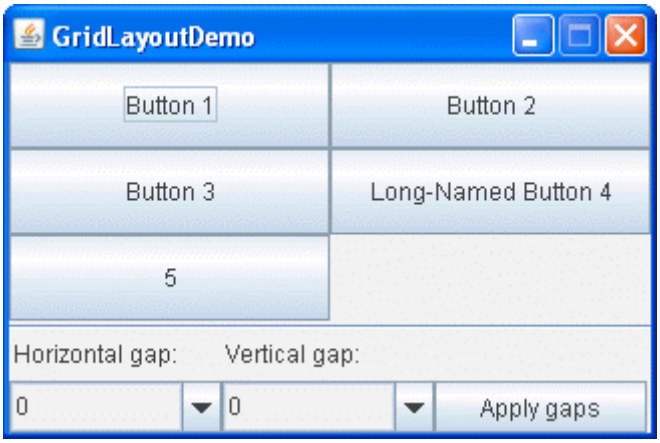


Η βασική κλάση του swing είναι η JFrame. Αυτή η κλάση είναι στην ουσία το κεντρικό παράθυρο της εφαρμογής. Χωρίς αυτό δεν μπορεί να υπάρξει εφαρμογή γραφικά απεικονισμένη. Μέσα στο JFrame μπορεί να χτιστούν όλα τα υπόλοιπα στοιχεία που χρειάζονται για μια εφαρμογή, όπως JPanels, JTextAreas και διάφορα άλλα

Το swing παρέχει μια ομάδα κλάσεων(Layout Managers) με τις οποίες γίνεται η διαχείριση της τοποθέτησης των γραφικών στοιχείων. Υπάρχουν διάφοροι τύποι Layout Managers όπως οι BorderLayout Manager, GridLayout, GroupLayout, BoxLayout και άλλοι. Κάθε ένας από αυτούς έχει διαφορετική φιλοσοφία για την τοποθέτηση των αντικειμένων. Για παράδειγμα ένα JPanel που έχει BorderLayout (panel.setLayoutManager(new BorderLayoutManager)) μπορεί να τοποθετήσει τα αντικείμενα μέσα σ'αυτό με γεωγραφικό τρόπο



ενώ ένα JPanel που έχει GridLayout (panel.setLayoutManager(new GridLayoutManager)) τοποθετεί τα επιπλέον γραφικά αντικείμενα σε ένα πλέγμα.



Listeners: διάδραση με τον χρήστη

Κάποια από τα αντικείμενα του swing μπορούν να επιστρέφουν ενέργειες που γίνονται σε αυτά από τον χρήστη όπως το πάτημα ενός JButton ή η μετακίνηση ενός JSlider. Για να επιστραφούν οι πληροφορίες της διάδρασης πρέπει να χρησιμοποιηθούν οι Listeners οι οποίοι παρέχονται από το πακέτο awt της Java. Υπάρχουν διαφορετικοί τύποι Listeners για τα διάφορα αντικείμενα του swing. Τα JButtons χρησιμοποιούν τους ActionListeners, τα JSliders χρησιμοποιούν τους ChangeListeners, ενώ υπάρχουν Listeners για οποιαδήποτε άλλη διάδραση που μπορεί να χρησιμοποιήσει κάποιος, για παράδειγμα για την διάδραση με το ποντίκι ενός υπολογιστή υπάρχει ο MouseListener.

Αφού κάποιος θέσει έναν Listener για οποιοδήποτε αντικείμενο (Jbutton.addActionListener(ActionEvent)) πρέπει να γράψει μια μέθοδο για να καθορίσει την ενέργεια που θα ακολουθήσει την πυροδότηση του Listener. Για κάθε είδος Listener η μέθοδος πρέπει να ονομάζεται ανάλογα, δηλαδή για ένα ActionListener πρέπει να γραφτεί μια μέθοδος με το όνομα actionPerformed και να έχει μεταβλητή ένα αντικείμενο τύπου ActionEvent. Για παράδειγμα:

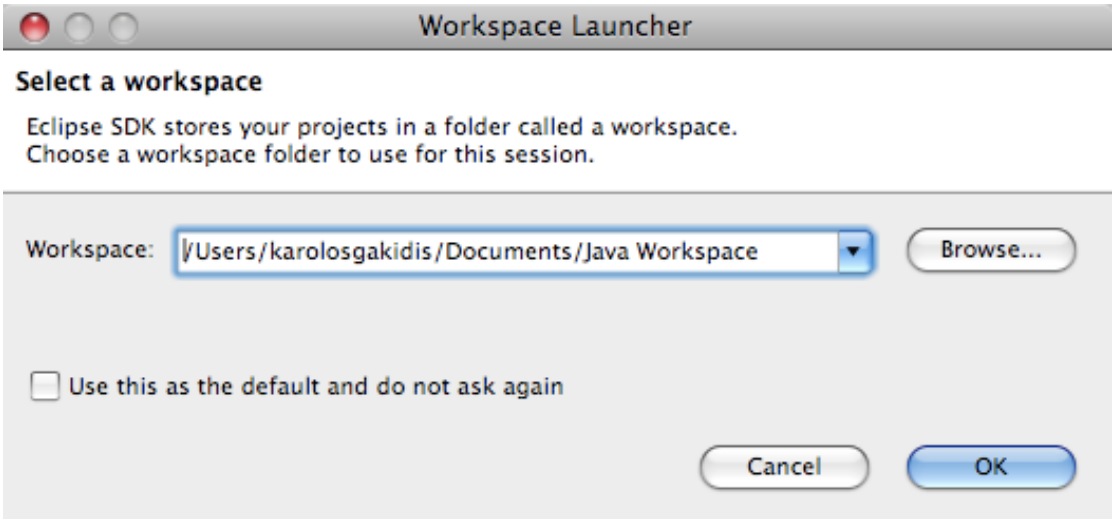
```
public void actionPerformed(ActionEvent e) {
            if (e.getSource() == queButton){
                  JButton source = (JButton) e.getSource();
                  if(source.isSelected()){
                  modul1.doQueue();
                  System.out.println("Note Played");
                  }
            }
```

Παράρτημα Β: Οδηγίες χρήσης

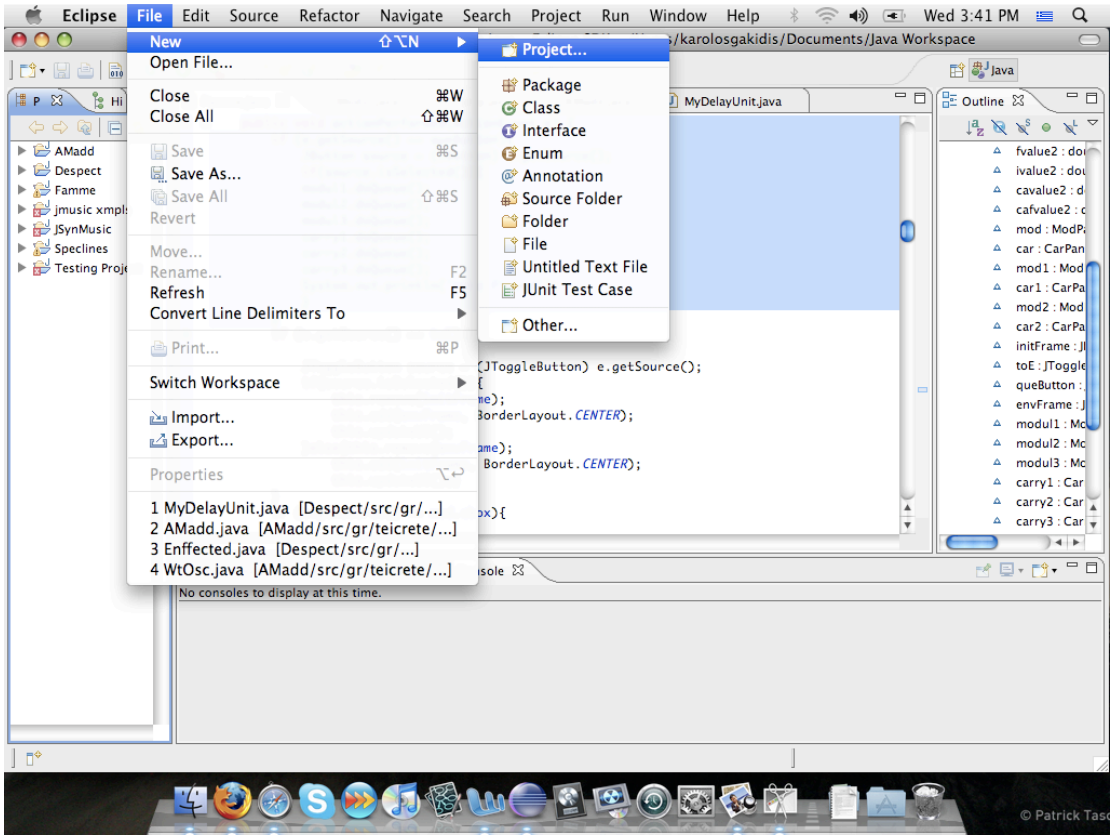
Εγκατάσταση του Jsyn API στο Eclipse SDK Δημιουργία εκτελέσιμου αρχείου jar

Η ανάπτυξη των εφαρμογών έγινε στο περιβάλλον Eclipse SDK. Είναι ένα περιβάλλον ανοιχτού κώδικα το οποίο διανέμεται δωρεάν στην ηλεκτρονική διεύθυνση www.eclipse.org και διευκολύνει τον προγραμματιστή παρέχοντας του εργαλεία για την ανάπτυξη εφαρμογών.

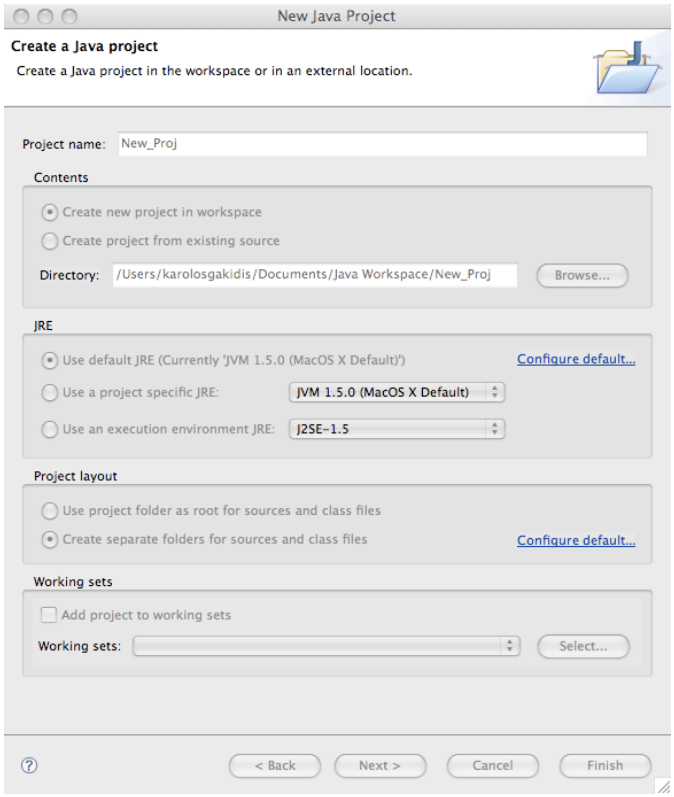
Για να ξεκινήσει κάποιος την ανάπτυξη εφαρμογών πρέπει αρχικά να δηλώσει τον φάκελο μέσα στον οποίο θα δουλεύει και θα γίνεται η διαχείριση των αρχείων του. Το eclipse ονομάζει αυτόν τον χώρο workspace και δίνει την δυνατότητα της επιλογής του χώρου αυτού κάθε φορά που εκκινείται η εφαρμογή.



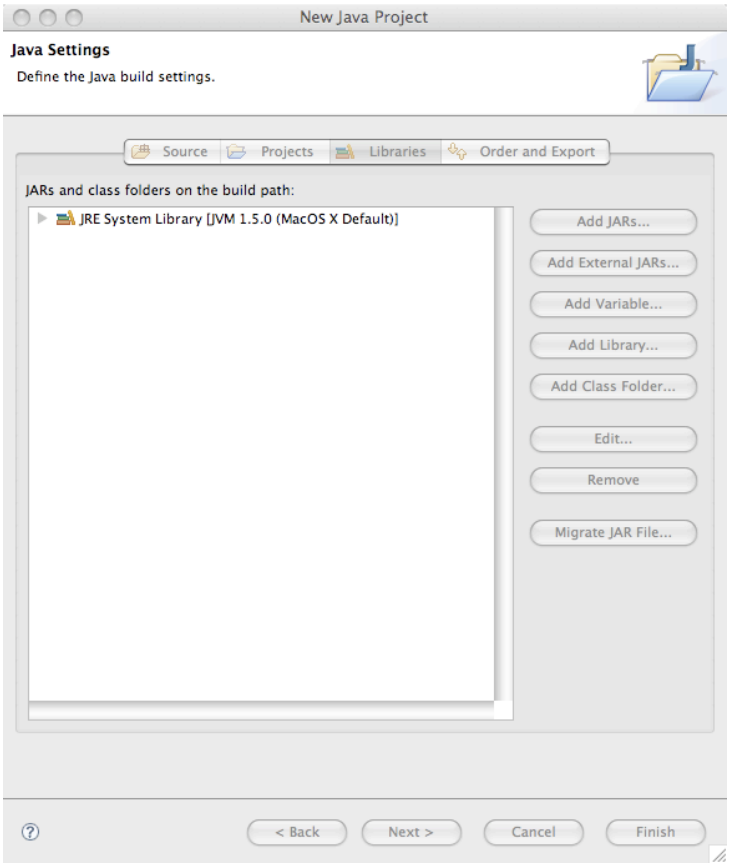
Αφού γίνουν όλα αυτά για να ξεκινήσει κάποιος να προγραμματίσει πρέπει να φτιάξει ένα eclipse project όπου θα υπάρχουν όλα τα σχετικά αρχεία με την εφαρμογή που θα φτιάξει.



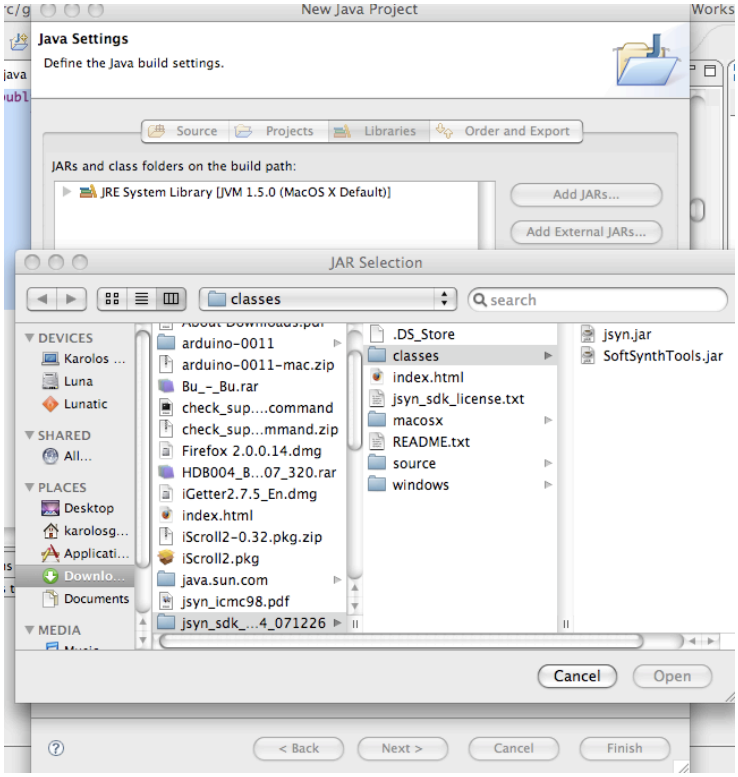
Στο καινούργιο αυτό project πρέπει αρχικά να επιλέξουμε το είδος του project. Επιλέγουμε λοιπόν "Java Project". Έπειτα πρέπει να του δώσουμε ένα όνομα και να δηλώσουμε τα εξωτερικές βιβλιοθήκες που θα χρησιμοποιήσουμε.



. Το επόμενο βήμα είναι η εγκατάσταση των βιβλιοθηκών του JSyn. Οι βιβλιοθήκες που χρειάζονται για να αναπτύξει κάποιος κώδικα με την χρήση του JSyn βρίσκονται στην ιστοσελίδα http://www.softsynth.com/jsyn/ developers/. Για να τις εγκαταστήσουμε επιλέγουμε την ταμπέλα Libraries και το κουμπί Add External Jars.

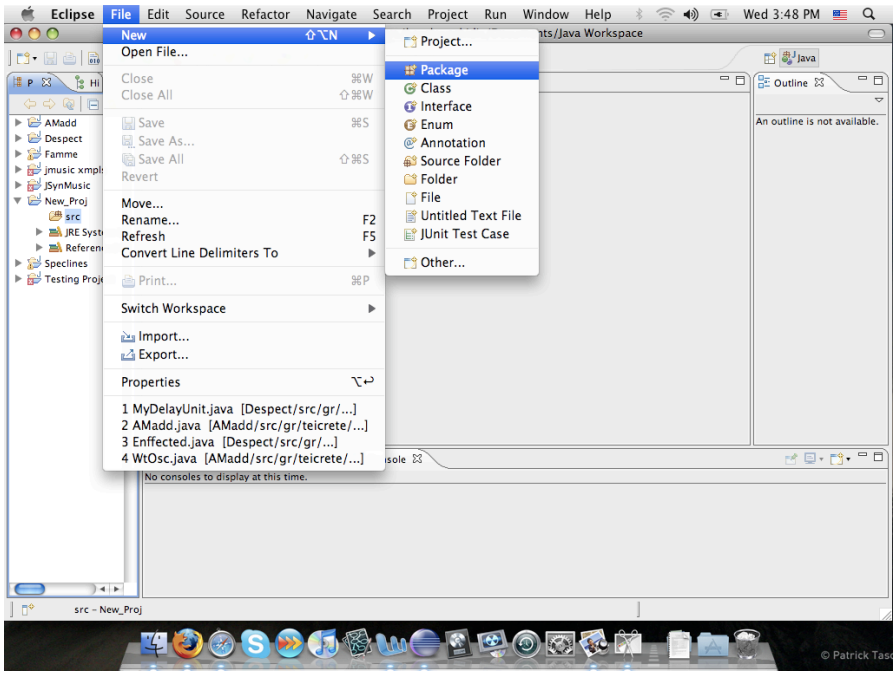


Θα πρέπει να επιλέξουμε δυο αρχεία jar για να έχουμε πλήρη πρόσβαση σε όλες τις κλάσεις του JSyn. Αυτά βρίσκονται στο φάκελο που μας παρέχει η ιστοσελίδα του JSyn μέσα στον υποφάκελο classes.



Επιλέγουμε λοιπόν τα δύο αρχεία jsyn.jar και SoftSynthTools.jar και πατάμε το κουμπί finish.

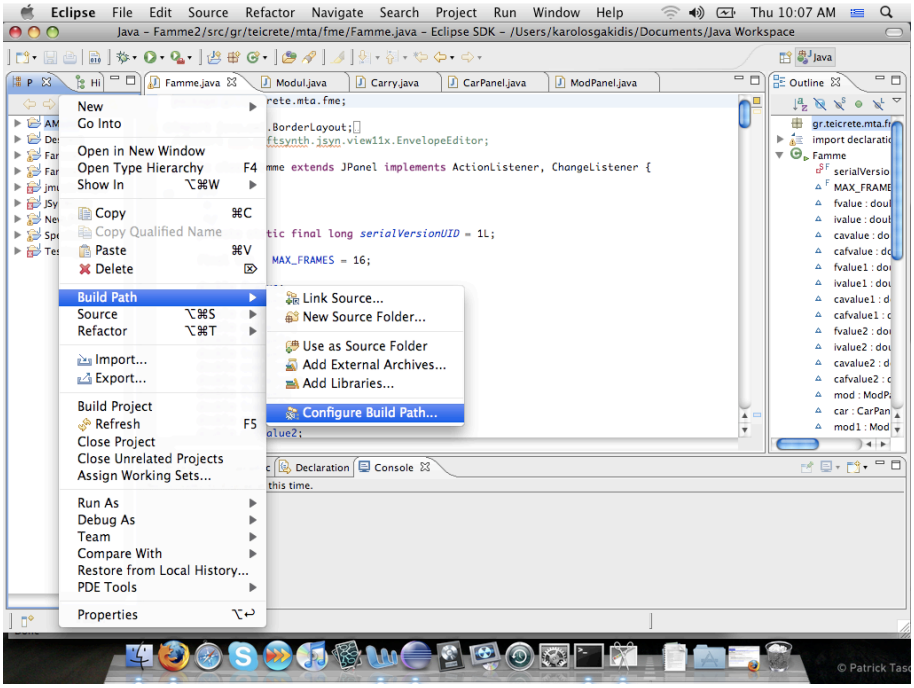
Για να ξεκινήσουμε τον προγραμματισμό στην Java είναι απαραίτητο να φτιάξουμε ένα νέο πακέτο μέσα στο project στο οποίο θα ανήκουν οι κλάσεις που θα φτιάξουμε. Έχοντας επιλεγμένο το καινούργιο project που μόλις φτιάξαμε επιλέγουμε File->New->Package και δίνουμε ένα όνομα στο πακέτο με μορφή για παράδειγμα gr.teicrete.mta.newpack



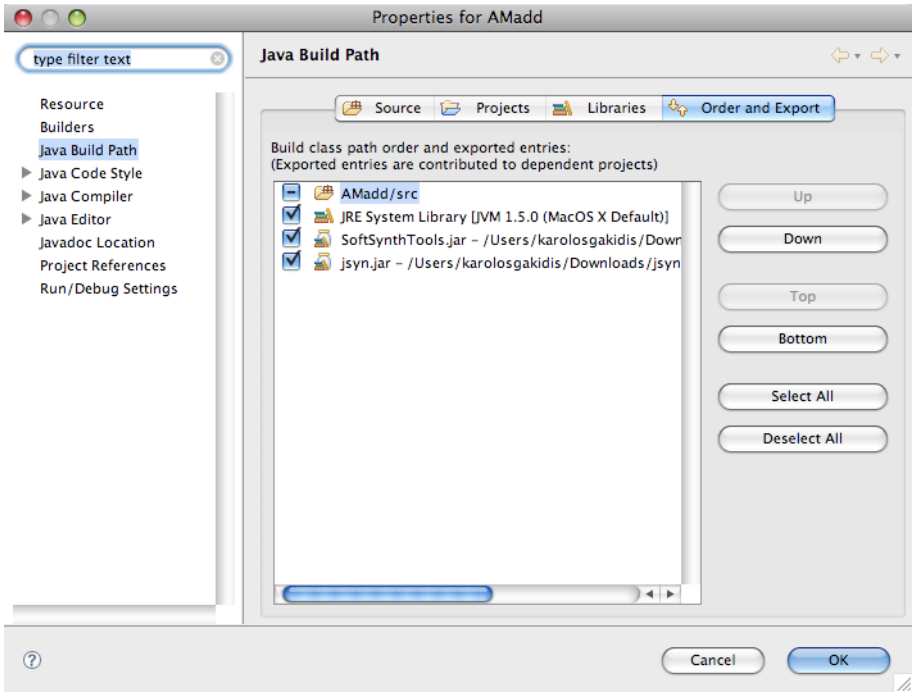
Τέλος φτιάχνουμε τις κλάσεις μας και αρχίζουμε τον προγραμματισμό. Πρέπει να σημειωθεί ότι τα ονόματα των κλάσεων πρέπει να ξεκινάνε με κεφαλαίο γράμμα. Έχοντας λοιπόν επιλεγμένο το πακέτο που μόλις φτιάξαμε πατάμε File->New->Class φτιάχνουμε μια νέα κλάση.

Δημιουργία εκτελέσιμου αρχείου jar-Εκτέλεση κώδικα

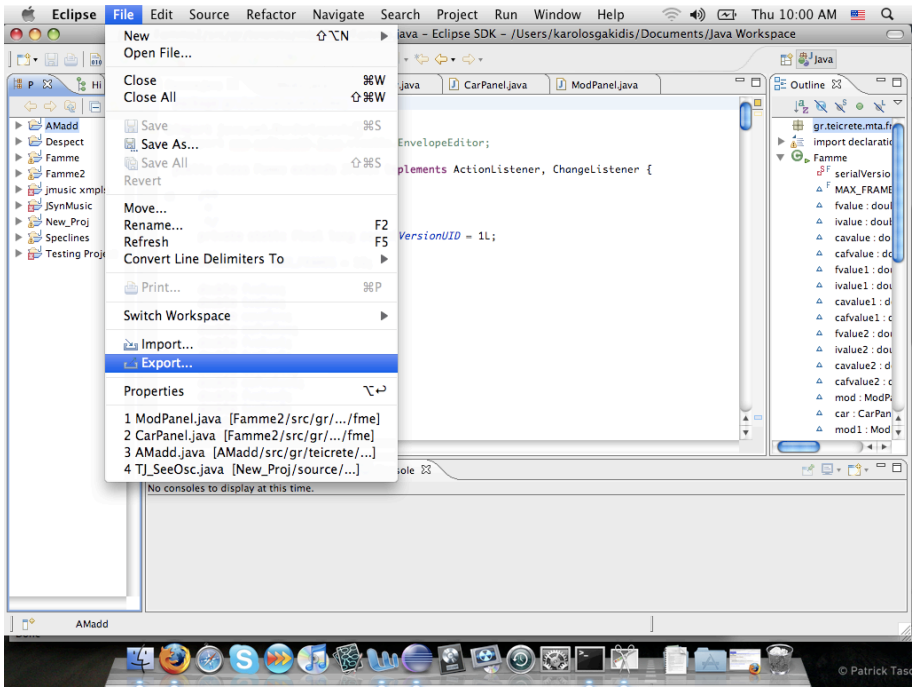
Για να δημιουργήσουμε ένα εκτελέσιμο αρχείο jar μας παρέχει το eclipse μια συλλογή από εργαλεία με τα οποία μπορεί να γίνει αυτό εύκολα και γρήγορα. Αρχικά πρέπει να ελέγξουμε τις βιβλιοθήκες που θα ενταχθούν μέσα στο εκτελέσιμο αρχείο και να επιλέξουμε να μπουν οι βιβλιοθήκες του JSyn. Με δεξί κλικ πάνω στο project που θέλουμε να εξάγουμε, βλέπουμε ένα μενού. Επιλέγουμε Build Path->Configure Build Path



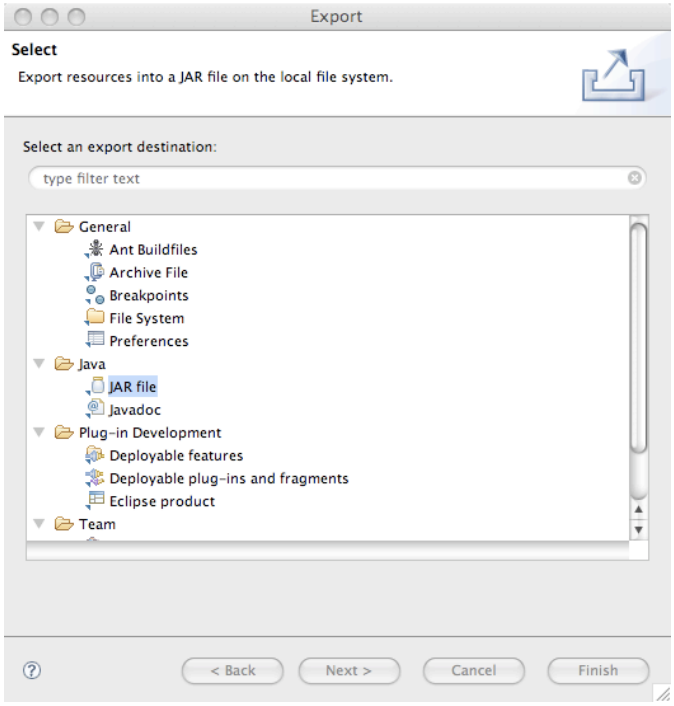
Εμφανίζεται τώρα ένα παράθυρο. Επιλέγουμε την ταμπέλα Order and Export. Φροντίζουμε να επιλέξουμε μαζί με όλες τις βιβλιοθήκες της Java να επιλέξουμε και τις βιβλιοθήκες του JSyn.



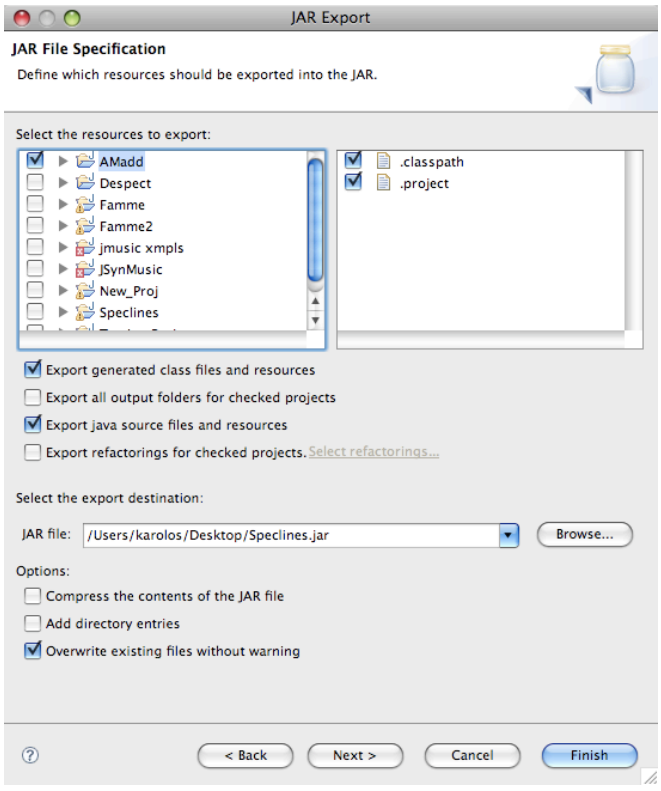
Για να δημιουργήσουμε το αρχείο τώρα πρέπει να επιλέξουμε File->Export.



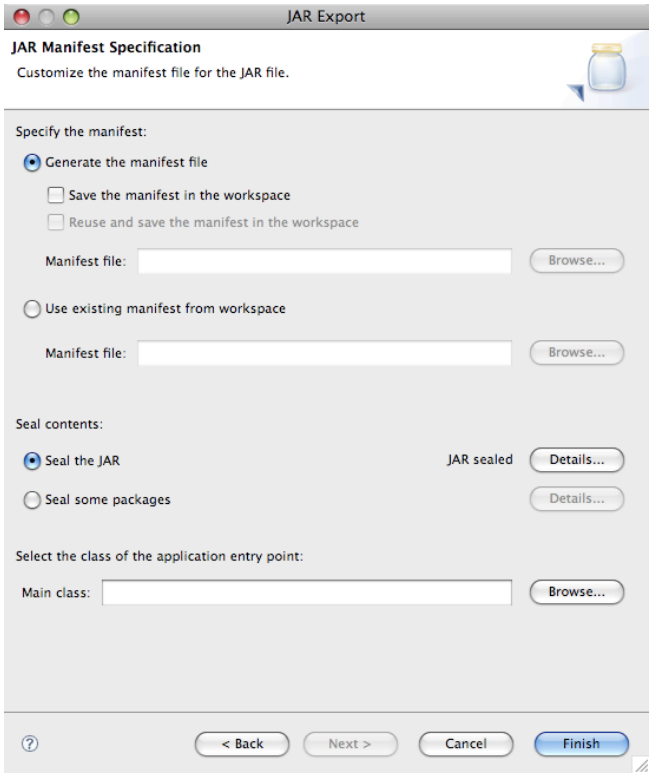
Το eclipse θα μας εμφανίσει ένα παράθυρο στο οποίο πρέπει να επιλέξουμε το είδος του αρχείου που θέλουμε να δημιουργήσουμε, το project από το οποίο θα γίνει η δημιουργία, την main κλάση του project και ένα manifest αρχείο το οποίο περιέχει τις πληροφορίες για τις βιβλιοθήκες που χρησιμοποιούμε. Το τελευταίο μπορεί να το δημιουργήσει αυτόματα το eclipse.



Επιλέγουμε λοιπόν Jar file και προχωράμε.



Εδώ επιλέγουμε το project και το που θέλουμε να σωθεί το jar αρχείο.



Τέλος επιλέγουμε την main κλάση και το αρχείο manifest.

Βιβλιογραφία

Java

Horton, I. (2005), 'Beginning In Java JDK 5', Wrox Press.

Eckel, B. (2002), 'Thinking In Java', Prentice-Hall.

JSyn

Burk, P.,(1998), 'JSyn – A Real-time Synthesis API for Java', ICMC Journal

Σύνθεση Ήχου

Διαμαντόπουλος, Τ.,(2004) 'Προγραμματισμός & Σύνθεση Ήχου', Έλλην

Rodes, C.,(1996) 'Computer Music Tutorial' , The MIT Press

Ηλεκτρονικές Διευθύνσεις

http://java.sun.com/j2se/1.4.2/docs/api/index.html (2008) Java Online Documentation

http://java.sun.com/docs/books/tutorial/index.html (2008) Java Online Tutorials

http://users.teicrete.gr/taxd/04/jnotes/j00.htm

http://www.didkovsky.com/JavaMusicSystems/JSyn1.pdf

http://www.softsynth.com/jsyn