Νικόλαος Παπαδακάκης

Πτυχιακή Εργασία

# Γραφιστικό Περιβάλλον Σύνθεσης Ήχου Συνδυαστικής Διαμόρφωσης AM, FM σε Πραγματικό Χρόνο

Επιβλέπων Καθηγητής
Ταξιάρχης Διαμαντόπουλος

Τ.Ε.Ι. Κρήτης
Παράρτημα Ρεθύμνου
Τμήμα Μουσικής Τεχνολογίας & Ακουστικής

Ρέθυμνο
Ιούνιος 2005

Περιεχόμενα

# Εισαγωγή

Σκοπός της εργασίας αυτής είναι η δημιουργία ενός γραφιστικού περιβάλλοντος σύνθεσης ήχου, το οποίο χρησιμοποιεί τις τεχνικές διαμόρφωσης πλάτους και διαμόρφωσης συχνότητας. Η εργασία αποτελείται από δύο σκέλη. Αφενός, εξερευνεί και εφαρμόζει τους αλγόριθμους διαμόρφωσης σήματος στη σύνθεση ήχου. Αφετέρου, χρησιμοποιεί την γλώσσα προγραμματισμού Java και εξερευνεί τις δυνατότητες που αυτή παρέχει σε σχέση με τη διαχείριση και την επεξεργασία του ηχητικού σήματος, καθώς και με τη δημιουργία γραφικού περιβάλλοντος ελέγχου.

Οι τεχνικές διαμόρφωσης σήματος χρησιμοποιούνται στις τηλεπικοινωνίες (ασύρματη μετάδοση σήματος) εδώ και πολλές δεκαετίες. Σαν περιβάλλοντα σύνθεσης ήχου πρωτοεξερευνήθηκαν ήδη από τις δεκαετίες του 1950, για τη μεν πρώτη, και του 1970, για την δεύτερη. Δημοφιλέστερη εφαρμογή τους στη σύνθεση ήχου, αποτελούν οι υποπεριπτώσεις του tremolo και του vibrato, αντίστοιχα. Οι τεχνικές αυτές αποτελούσαν για μία μεγάλη χρονική περίοδο μία σχετικά χαμηλού κόστους λύση για τα περιβάλλοντα σύνθεσης ήχου, σε εποχές όπου το κόστος υλικού ήταν απαγορευτικό. Έτσι, με μικρό πλήθος ταλαντωτών μπορούσε συχνά να παραχθεί ένα σήμα με πλούσιο συχνοτικό φάσμα. Ταυτόχρονα, οι τεχνικές αυτές προσδίδουν ένα ιδιαίτερο ηχοχρωματικό περιεχόμενο, ενώ κατά καιρούς πραγματοποιήθηκαν και σχετικές έρευνες -με επιτυχή συχνά αποτελέσματα- προκειμένου να αναπαραχθούν φυσικά μουσικά όργανα.

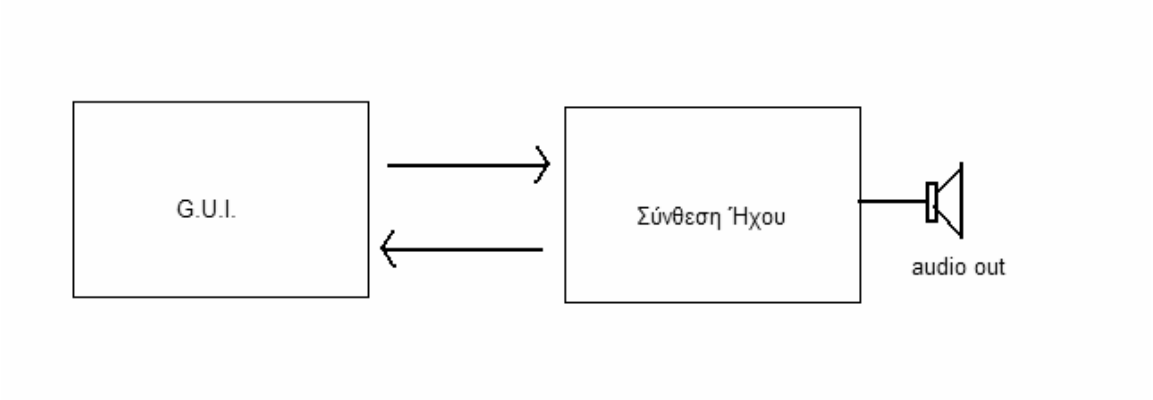
Ο κώδικας για τη δημιουργία του περιβάλλοντος αυτού αναπτύχθηκε με το Java 2 Software Development Kit (Standard Edition , Version 1.4.02). Η Java είναι μια αντικειμενοστραφής, ανεξάρτητης πλατφόρμας, γλώσσα προγραμματισμού. Ο όρος ‘ανεξάρτητης πλατφόρμας’ σημαίνει πως μία εφαρμογή που αναπτύσσεται σε Java, μπορεί να εκτελείται σε διαφορετικά περιβάλλοντα υπολογιστών χωρίς τροποποίηση. Αυτό συμβαίνει γιατί η ίδια η γλώσσα είναι εφοδιασμένη με ειδικές βιβλιοθήκες, οι οποίες κατά τη διάρκεια εκτέλεσης του κώδικα αναλαμβάνουν την μετατροπή των εντολών του, στις αντίστοιχες εντολές του περιβάλλοντος εκτέλεσής του (στο αντίστοιχο δηλαδή hardware και λειτουργικό σύστημα). Ένα δεύτερο χαρακτηριστικό της γλώσσας αυτής, είναι η ανοικτή αρχιτεκτονική της. Αυτό έχει σαν αποτέλεσμα διάφοροι προγραμματιστές της Java να επεκτείνουν και να εμπλουτίζουν διαρκώς τη γλώσσα με νέες δυνατότητες. Από το 1995 που κυκλοφόρησε η πρώτη έκδοση της Java μέχρι σήμερα, έχουν δημιουργηθεί διάφορες δικτυακές κοινότητες στις οποίες ανταλλάσσεται ελεύθερα κώδικας και απόψεις σχετικά με την γλώσσα και την εξέλιξη της. Επίσης, αντίθετα από άλλες γλώσσες, η δομή της Java επιτρέπει τον σχεδιασμό μιας εφαρμογής με βάση την αφαιρετική προσέγγιση (abstract programming). Αυτό έχει σαν αποτέλεσμα, τα διάφορα προγραμματιστικά αντικείμενα που δημιουργούνται για μια συγκεκριμένη εφαρμογή, να μπορούν να ξαναχρησιμοποιηθούν αυτούσια στο μέλλον, από μία διαφορετική εφαρμογή. Τέλος, ένα επιπλέον πλεονέκτημα, αφορά στο γεγονός πως οι διάφορες εκδόσεις της γλώσσας διατίθενται ελεύθερα από την εταιρεία Sun. Τα παραπάνω χαρακτηριστικά έχουν οδηγήσει στο να προσεγγιστεί η Java από ένα μεγάλο πλήθος προγραμματιστών και εκπαιδευτικών ιδρυμάτων. Είναι επίσης οι ίδιοι λόγοι που οδήγησαν στην επιλογή της Java για την εκπόνηση της εργασίας αυτής.

Ως προς την ίδια την εφαρμογή, πρέπει να επισημανθεί πως σκοπός της εργασίας δεν αποτελεί τόσο η μουσική δημιουργία, αλλά κυρίως η δημιουργία ενός software synthesizer, το οποίο θα εκμεταλλεύεται όσο το δυνατόν περισσότερο τις συγκεκριμένες τεχνικές σύνθεσης ήχου. Έτσι, εκτός από την απλή εφαρμογή των τεχνικών αυτών, δίνεται η δυνατότητα συνδυαστικών διατάξεων (multiple carrier και multiple modulator, εν σειρά ή παράλληλα), ενώ διερευνήθηκε και η δυνατότητα ταυτόχρονης ύπαρξης AM και FM στο ίδιο κύκλωμα πολλαπλής διαμόρφωσης. Μεγάλο ενδιαφέρον είχε επίσης και η διερεύνηση των δυνατοτήτων της γλώσσας Java σε σχέση με τις audio εφαρμογές και ιδιαίτερα με την αποτελεσματικότητά της σε αλγόριθμους σύνθεσης ήχου. Για τον λόγο αυτόν η εργασία περιλαμβάνει μία σειρά από μετρήσεις που έγιναν με σκοπό την πειραματική επαλήθευση των αλγορίθμων που χρησιμοποιήθηκαν.

# 1. Περιγραφή του Περιβάλλοντος AudioSynth

## 1.1 Γενικά Χαρακτηριστικά

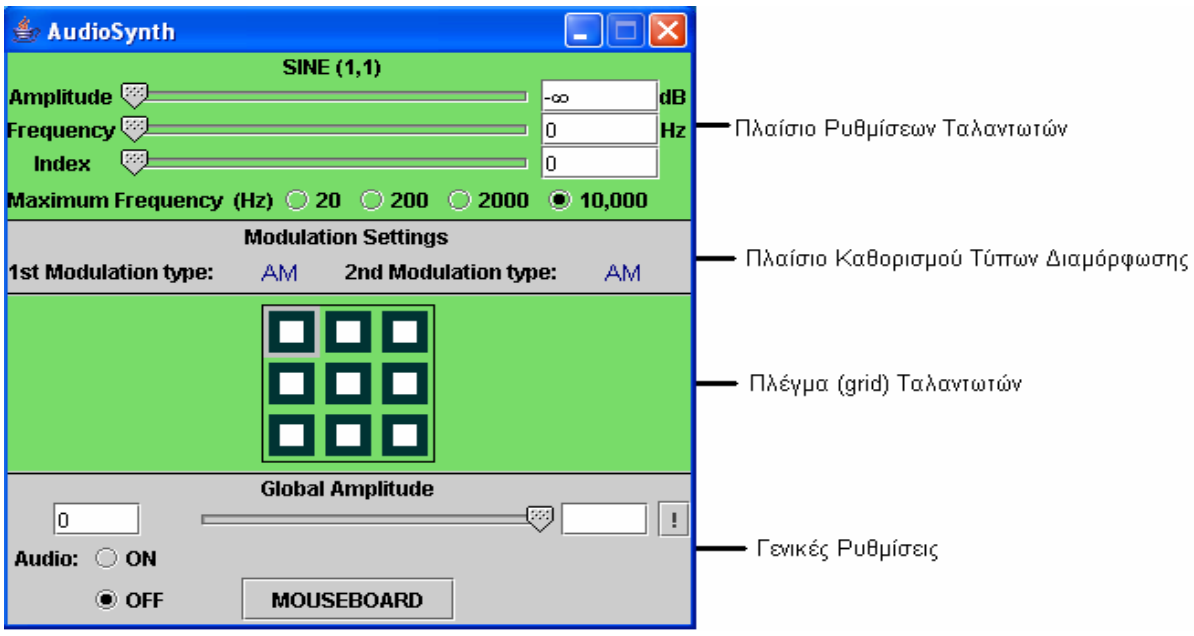
Η εφαρμογή AudioSynth αποτελείται από το γραφιστικό User Interface και από τους αλγόριθμους σύνθεσης ήχου. Μέσα από το user interface ο χρήστης έχει τη δυνατότητα να δημιουργήσει διάφορα κυκλώματα σύνθεσης ήχου και να επέμβει στις παραμέτρους τους σε πραγματικό χρόνο. Αυτό συμβαίνει με τη χρήση των γραφιστικών στοιχείων του interface, η κατάσταση των οποίων ορίζει τις διάφορες παραμέτρους των αλγόριθμων σύνθεσης ήχου. Το παρακάτω σχήμα απεικονίζει αυτήν την επικοινωνία μεταξύ των δύο βασικών μερών της εφαρμογής.




Εικόνα 1.1

## 1.2 Περιγραφή του User Interface

Το κύριο παράθυρο του GUI της εφαρμογής AudioSynth έχει ως εξής:




Εικόνα 1.2

Στην παραπάνω εικόνα φαίνεται το αρχικό παράθυρο του AudioSynth το οποίο εμφανίζεται στην επιφάνεια εργασίας του χρήστη μόλις αυτός εκτελέσει την εφαρμογή. Είναι εμφανές ότι αυτό το παράθυρο που αποτελεί και το κύριο παράθυρο της εφαρμογής μπορεί να χωριστεί στα εξής τέσσερα πλαίσια:

- πλαίσιο ρυθμίσεων ταλαντωτών,
- πλαίσιο καθορισμού τύπων διαμόρφωσης,
- grid των ταλαντωτών και
- πλαίσιο γενικών ρυθμίσεων.

Η περιγραφή των πλαισίων αυτών, γίνεται με βάση την πιθανή σειρά των ενεργειών που πρέπει να ακολουθήσει ο χρήστης για να χρησιμοποιήσει την εφαρμογή.

### 1.2.1 Πλέγμα (Grid) Ταλαντωτών)

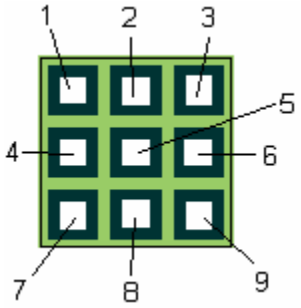
Στο πλαίσιο του πλέγματος ταλαντωτών ο χρήστης μπορεί να ενεργοποιήσει ή να απενεργοποιήσει εννέα ημιτονικούς ταλαντωτές. Γενικά, προκειμένου ο χρήσης να υλοποιήσει κάποιο κύκλωμα σύνθεσης ήχου χρειάζεται:

α. να ενεργοποιήσει συγκεκριμένο πλήθος ταλαντωτών μέσω του πλαισίου πλέγματος (grid), και
β. να καθορίσει τους τύπους διαμόρφωσης που θα εφαρμοστούν.

Το grid των ταλαντωτών έχει σχεδιαστεί ώστε να μιμείται αντίστοιχα grid παλαιών αναλογικών synthesizer όπως του VCS 3 (Εικόνα 1.3):

Εικόνα 1.3

Το grid μπορούμε να το φανταστούμε σαν έναν 3x3 πίνακα από ON/OFF διακόπτες. Στην παρακάτω εικόνα (Εικόνα 1.4) βλέπουμε το πλέγμα των ταλαντωτών με αριθμημένο τον κάθε διακόπτη του. Αυτήν την αρίθμηση θα χρησιμοποιήσουμε στο επόμενο κεφάλαιο κατά την εξέταση των διαφόρων πιθανών καταστάσεων του grid.



Εικόνα 1.4

Στην εικόνα 1.2 όλοι οι διακόπτες του πλέγματος ταλαντωτών είναι σε κατάσταση OFF και επομένως δεν υλοποιείται κανένα κύκλωμα. Για να θέσει ο χρήστης σε κατάσταση ON έναν από τους διακόπτες του πλέγματος ταλαντωτών πρέπει να κάνει αριστερό κλικ επάνω του. Σε αυτήν την περίπτωση θα εμφανιστεί ένας μικρός κύκλος εντός του πλαισίου του αντίστοιχου διακόπτη που υποδεικνύει ότι είναι σε κατάσταση ON:


Εικόνα 1.5

### 1.2.2 Πλαίσιο Καθορισμού Τύπων Διαμόρφωσης

Όπως δηλώνει και το όνομα του, στο συγκεκριμένο πλαίσιο καθορίζουμε τους τύπους των διαμορφώσεων που θα υλοποιηθούν στο κύκλωμα σύνθεσης ήχου. Στην παρακάτω εικόνα βλέπουμε όλες τις πιθανές καταστάσεις που μπορεί να έχει το πλαίσιο αυτό.

| Modulation Settings | | | |
|---|---|---|---|
| 1st Modulation type: | AM | 2nd Modulation type: | AM |
| **Modulation Settings** | | | |
| 1st Modulation type: | AM | 2nd Modulation type: | FM |
| **Modulation Settings** | | | |
| 1st Modulation type: | FM | 2nd Modulation type: | AM |
| **Modulation Settings** | | | |
| 1st Modulation type: | FM | 2nd Modulation type: | FM |

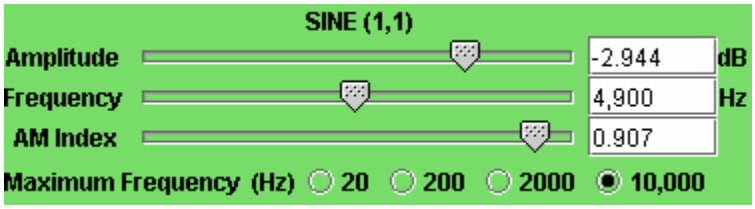
Εικόνα 1.6

Οι ρυθμίσεις του πλαισίου καθορισμού των τύπων διαμόρφωσης της παραπάνω εικόνας σε συνδυασμό με τις πιθανές καταστάσεις των διακοπτών του grid, ορίζουν το κύκλωμα σύνθεσης ήχου το οποίο υλοποιείται.

### 1.2.3 Πλαίσιο Ρυθμίσεων Ταλαντωτών

Το πλαίσιο ρυθμίσεων ταλαντωτών αφορά στις ρυθμίσεις καθενός από τους εννέα διαφορετικούς ημιτονικούς ταλαντωτές του AudioSynth. Οι ρυθμίσεις για τον κάθε ταλαντωτή διαφέρουν ανάλογα με την κατάσταση του grid. Αυτό συμβαίνει γιατί χρειάζεται να έχουμε τη δυνατότητα διαφορετικών ρυθμίσεων για έναν ταλαντωτή όταν αυτός είναι διαμορφωτής πλάτους, διαμορφωτής συχνότητας, φορέας ή απενεργοποιημένος.

Στην εικόνα 1.7 βλέπουμε ένα παράδειγμα ρυθμίσεων του πλαισίου ρυθμίσεων ταλαντωτών όταν αυτό αναφέρεται στον ταλαντωτή 1:



Εικόνα 1.7

Παρατηρούμε καταρχήν την ετικέτα 'SINE (1,1)' στην κορυφή του πλαισίου. Αυτή μας ενημερώνει ότι το πλαίσιο αφορά στις ρυθμίσεις του ταλαντωτή που αντιστοιχεί στον διακόπτη που βρίσκεται στην πρώτη γραμμή και πρώτη στήλη του πλέγματος ταλαντωτών, αφορά δηλαδή στις ρυθμίσεις του ταλαντωτή 1 σύμφωνα με την αρίθμηση που δώσαμε προηγουμένως.

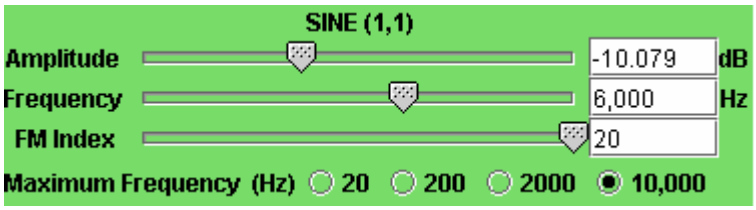
Επίσης παρατηρούμε τρία sliders τα οποία συνοδεύονται από επεξηγηματικές ετικέτες και πλαίσια εμφάνισης τιμών. Το πρώτο slider αφορά, όπως φαίνεται και από την ετικέτα που βρίσκεται αριστερά του, στο πλάτος του ταλαντωτή. Προγραμματιστικά, το πλάτος ενός ταλαντωτή παίρνει τιμές από 0 ως 1. Ωστόσο, στο πλαίσιο εμφάνισης των τιμών του πλάτους, αυτές εμφανίζονται ως λόγος dB. Έτσι όταν το πλάτος του ταλαντωτή είναι 0 στο αντίστοιχο πλαίσιο θα εμφανίζεται η τιμή -∞ dB ενώ όταν το πλάτος του ταλαντωτή είναι 1 θα εμφανίζεται η τιμή των 0 dB.

Κάτω από το slider του πλάτους βρίσκεται το slider της συχνότητας το οποίο συνοδεύεται επίσης από ένα πλαίσιο εμφάνισης τιμών. Στο πλαίσιο αυτό εμφανίζεται η τιμή της συχνότητας ταλαντωτή σε Hz. Το εύρος των συχνοτήτων που μπορούμε να ρυθμίσουμε σε κάθε ταλαντωτή, προσδιορίζεται από την ομάδα των τεσσάρων επιλογών για τη μέγιστη συχνότητα του ταλαντωτή, που βρίσκεται στο κάτω μέρος του πλαισίου ρυθμίσεων ταλαντωτή. Με αυτές τις τέσσερις επιλογές ο χρήστης μπορεί να επιλέξει τιμές συχνότητας από 0 ως 20 Hz, από 0 ως 200 Hz , από 0 ως 2000Hz και από 0 ως 10000 Hz. Η προσθήκη αυτής της ομάδας επιλογών έγινε προκειμένου ο χρήστης να έχει ένα είδος 'fine tuning', δηλαδή καλύτερο έλεγχο και μεγαλύτερη ακρίβεια στις χαμηλές συχνότητες, χαρακτηριστικό ιδιαίτερα χρήσιμο στην επιλογή τιμών για έναν ταλαντωτή διαμόρφωσης. Το πλήθος των ενδιάμεσων τιμών που μπορεί να αναπαράγει κάθε slider είναι ενός πεπερασμένου αριθμού, με αποτέλεσμα να μην είναι δυνατό να παραχθούν κάποιες ενδιάμεσες συχνότητες. Έτσι, μειώνοντας την μέγιστη δυνατή συχνότητα στα 20 Hz μπορούμε να έχουμε ακρίβεια της τάξης των 0,1 Hz. Φυσικά, οι μικρές αυτές συχνότητες και οι αποκλίσεις τέτοιας τάξης δεν είναι ακουστές στην περίπτωση που ο

ταλαντωτής είναι φορέας. Στην περίπτωση όμως που είναι διαμορφωτής πλάτους ή συχνότητας, η συχνότητα του αποτελεί την ταχύτητα διαμόρφωσης (modulation rate) και επομένως οι συχνότητες αυτές και η ακρίβεια αυτή μπορεί να είναι επιθυμητές.

Το τρίτο slider αφορά στον συντελεστή διαμόρφωσης. Ο συντελεστής αυτός στην περίπτωση της AM διαμόρφωσης μπορεί να πάρει τιμές από 0 ως 1. Οι τιμές αυτές θα εμφανίζονται στο πλαίσιο εμφάνισης τιμών που βρίσκεται δεξιά του slider.

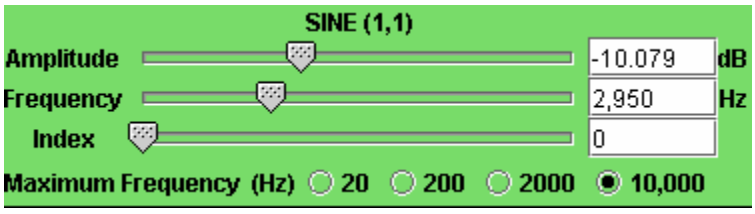
Αν κάνουμε τις απαραίτητες ρυθμίσεις ώστε ο ταλαντωτής 1 να είναι διαμορφωτής συχνότητας το πλαίσιο ρυθμίσεων του θα έχει τη μορφή της εικόνας 1.8:



Εικόνα 1.8

Από το σχήμα παρατηρούμε πως, η μόνη διαφορά στο πλαίσιο ρυθμίσεων ταλαντωτή αφορά στον συντελεστή διαμόρφωσης που όπως φαίνεται και στην σχετική ετικέτα τώρα αναφέρεται σε FM διαμόρφωση. Επίσης, ο slider για τον συντελεστή διαμόρφωσης έχει πάρει τη μέγιστη τιμή του. Η μέγιστη τιμή για τον συντελεστή διαμόρφωσης συχνότητας έχει οριστεί να είναι το 20, όπως φαίνεται και στο αντίστοιχο πεδίο εμφάνισης τιμών. Η τιμή αυτή έχει οριστεί με βάση αισθητικά κριτήρια σχετικά με το ηχητικό αποτέλεσμα.

Ένα ακόμα παράδειγμα αφορά στη περίπτωση ενός ταλαντωτή-φορέα όπως φαίνεται και στο παρακάτω σχήμα:

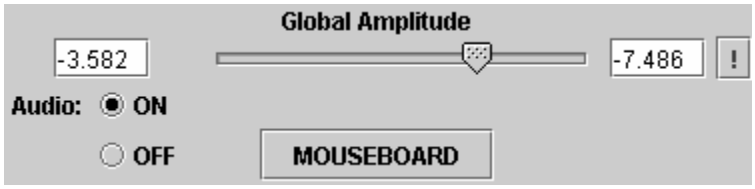


Εικόνα 1.9

Όπως βλέπουμε, το πλαίσιο ρυθμίσεων ταλαντωτή έχει πανομοιότυπη μορφή με τις δύο προηγούμενες περιπτώσεις. Φυσικά, οι όποιες ρυθμίσεις του slider με την ετικέτα ¨Index¨ έχουν νόημα μόνο για την περίπτωση που ο χρήστης πρόκειται να δώσει στον συγκεκριμένο ταλαντωτή την ιδιότητα του διαμορφωτή.

### 1.2.4 Πλαίσιο Γενικών Ρυθμίσεων

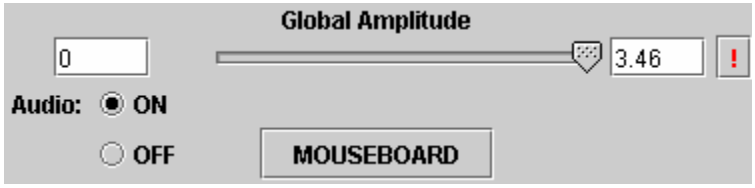
Το πλαίσιο του κύριου παράθυρου του AudioSynth που μένει να εξετάσουμε, είναι αυτό των γενικών ρυθμίσεων (Εικόνα 1.10).



Εικόνα 1.10

Βλέπουμε καταρχήν ένα ρυθμιστικό με την ετικέτα 'Global Amplitude'. Όπως υποδηλώνει και η ετικέτα του, το slider αυτό ρυθμίζει την συνολική ένταση του AudioSynth. Στο αριθμητικό πλαίσιο που βρίσκεται στα αριστερά του slider εμφανίζεται η τιμή που αφορά στην ρύθμιση της συνολικής έντασης στην έξοδο της εφαρμογής, με εύρος τιμών μεταξύ 0dB (μέγιστη) και -∞ (ελάχιστη). Στο αριθμητικό πλαίσιο που βρίσκεται δεξιά από το slider εμφανίζεται η τρέχουσα μέγιστη στάθμη που έχει αναπαραχθεί στην έξοδο, δυνατότητα ιδιαίτερη χρήσιμη ώστε να γνωρίζουμε εάν το κύκλωμα διαμόρφωσης θα παραμορφώσει στην έξοδο. Προκειμένου να μηδενιστεί η συγκεκριμένη μέτρηση (reset), ο χρήστης θα πρέπει να κάνει κλικ πάνω στο κουμπί με το θαυμαστικό που βρίσκεται δεξιά του πλαισίου.

Στην εικόνα 1.10 βλέπουμε ότι ο χρήστης έχει υποβαθμίσει το σήμα εξόδου κατά 3.6 περίπου dB και η έξοδος του οργάνου είναι στα -7.48 dB. Αν η έξοδος του οργάνου υπερβεί τα 0 dB θα έχουμε παραμόρφωση του σήματος και το πλαίσιο γενικών ρυθμίσεων θα πάρει την παρακάτω μορφή:



Εικόνα 1.11

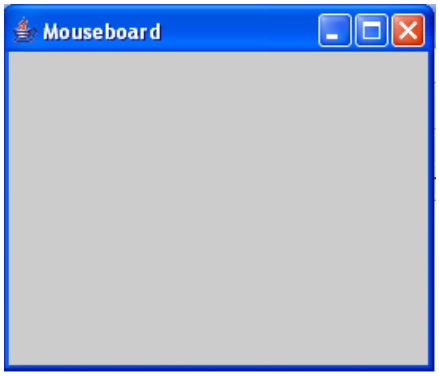
Όπως βλέπουμε τώρα το σήμα εξόδου είναι στα +3,46 dB και προφανώς έχουμε παραμόρφωση. Σε αυτήν την περίπτωση το θαυμαστικό στο κουμπί που χρησιμοποιείται ως reset για τις ενδείξεις του πλαισίου εμφάνισης της στάθμης εξόδου, γίνεται κόκκινο ώστε να προειδοποιείται ο χρήστης. Όταν ο χρήστης χαμηλώσει το πλάτος εξόδου για να αποφύγει την παραμόρφωση, θα πρέπει να ξαναπατήσει το reset ώστε να δει τη νέα στάθμη εξόδου. Αν αυτή είναι κάτω από τα 0 dB το θαυμαστικό θα επανέρθει στο αρχικό γκρι χρώμα του, αλλιώς θα παραμείνει κόκκινο.

Κάτω και αριστερά από το slider του συνολικού πλάτους του AudioSynth βλέπουμε τις επιλογές Audio ON και Audio OFF. Όπως είναι προφανές για να παράγεται οποιοσδήποτε ήχος από το AudioSynth θα πρέπει να τεθεί σε κατάσταση Audio ON.

Το τελευταίο συστατικό του πλαισίου γενικών ρυθμίσεων είναι ένα κουμπί με την ετικέτα ¨MOUSEBOARD¨. Κάνοντας κλικ ο χρήστης σε αυτό το κουμπί εμφανίζεται στην οθόνη του ένα νέο ανεξάρτητο παράθυρο τη χρησιμότητα του οποίου εξετάζουμε παρακάτω.

### 1.2.5 MOUSEBOARD

Στην εικόνα 1.12 φαίνεται το παράθυρο με τον τίτλο ¨Mouseboard¨ το οποίο ολοκληρώνει το γραφικό interface του AudioSynth.



Εικόνα 1.12

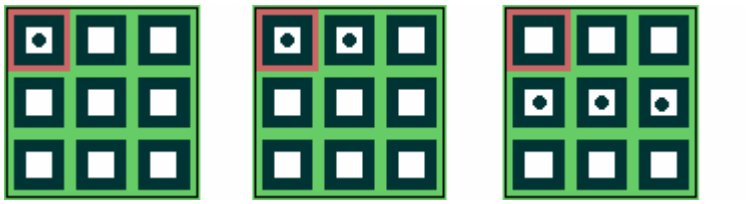
Όπως βλέπουμε πρόκειται για ένα παράθυρο που δεν περιέχει κανένα συστατικό. Η χρησιμότητα αυτού του παραθύρου είναι το ότι η κίνηση με το ποντίκι μέσα στα όρια του, αποτελεί έναν επιπλέον τρόπο ρύθμισης του πλάτους και της συχνότητας των ταλαντωτών. Ανάλογα τον διακόπτη ο οποίος έχει επιλεγεί εντός του πλέγματος ταλαντωτών, η κίνηση του κέρσορα εντός των ορίων του mouseboard αλλάζει τις ρυθμίσεις του αντίστοιχου ταλαντωτή. Η κίνηση στον οριζόντιο άξονα αφορά στην συχνότητα του ταλαντωτή και η κίνηση στον κάθετο το πλάτος ταλάντωσης, με αρχή αξόνων την κάτω αριστερή γωνία. Σημειώνεται, πως η κίνηση στον κάθετο άξονα προκαλεί γραμμική μεταβολή του πλάτους, ενώ η κίνηση στον οριζόντιο άξονα προκαλεί εκθετική μεταβολή της συχνότητας. Γενικά, οι τιμές πλάτους έχουν εύρος από $-\infty$ ως 0 dB, ενώ η μέγιστη συχνότητα που μπορούμε να παράγουμε εξαρτάται και πάλι από τη σχετική ρύθμιση στο πλαίσιο ρυθμίσεων ταλαντωτή. Επιπρόσθετα, καθώς ο κέρσορας κινείται μέσα στο mouseboard, μεταβάλλονται και οι αντίστοιχες ρυθμίσεις του πλαισίου ρυθμίσεων ταλαντωτή, δηλαδή τα slider του πλάτους και της συχνότητας καθώς και τα αντίστοιχα πεδία εμφάνισης τιμών.

# 2. Υλοποίηση των Κυκλωμάτων Σύνθεσης Ήχου

Όπως παρουσιάστηκε στο προηγούμενο κεφάλαιο, η μορφή των κυκλωμάτων σύνθεσης ήχου που υλοποιούνται στην εφαρμογή AudioSynth, ορίζονται από τις ρυθμίσεις του χρήστη στο πλαίσιο του πλέγματος ταλαντωτών και στο πλαίσιο καθορισμού τύπων διαμόρφωσης. Στο κεφάλαιο αυτό παρουσιάζονται οι τεχνικές σύνθεσης ήχου που μπορούν να υλοποιηθούν από την εφαρμογή, μαζί με τις αντίστοιχες ρυθμίσεις.

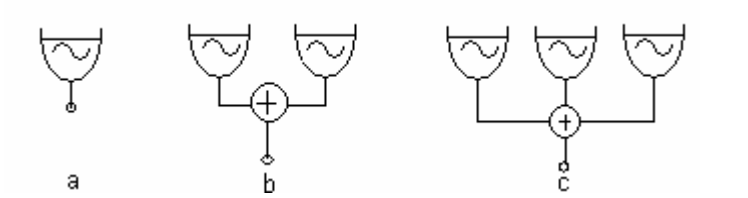
## 2.1 Προσθετική Σύνθεση Ταλαντωτών

Η πιο απλή περίπτωση αφορά στην απουσία διαμόρφωσης γενικότερα. Η υλοποίηση αυτή αφορά δηλαδή στην τεχνική της **Προσθετικής Σύνθεσης**. Έτσι σύμφωνα με το κύκλωμα αυτό, ένας ή περισσότεροι διακόπτες μιας μόνο σειράς, βρίσκονται σε κατάσταση ON:



Εικόνα 2.1

Τα κυκλώματα που αντιστοιχούν στις τρεις παραπάνω καταστάσεις του πλέγματος των ταλαντωτών φαίνονται στα τρία παρακάτω block διαγράμματα:




Εικόνα 2.2

Από το διάγραμμα a φαίνεται πως όταν τεθεί σε κατάσταση ON ένας μόνο από του διακόπτες του πλέγματος, αυτό που θα έχουμε στην έξοδο του ταλαντωτή είναι η έξοδος του αντίστοιχου ημιτονικού ταλαντωτή. Από τα διαγράμματα b και c παρατηρούμε ότι αν θέσουμε σε κατάσταση ON παραπάνω από έναν ταλαντωτή σε μια σειρά και δεν έχουμε θέσει σε κατάσταση ON κάποιον από τους ταλαντωτές των υπόλοιπων γραμμών του πλέγματος, στην έξοδο του grid θα έχουμε την πρόσθεση των ημιτονικών σημάτων των αντίστοιχων κάθε φορά ταλαντωτών. Επίσης από τα διαγράμματα b και c είναι εμφανές πως εάν έχουμε ενεργοποιημένους ταλαντωτές μόνο σε μια από τις τρεις σειρές του grid, δεν έχει σημασία ποια θα είναι αυτή.

Αν παρατηρήσουμε όλες τις παραπάνω εικόνες στις οποίες απεικονίζεται το πλέγμα ταλαντωτών, θα δούμε ένα κόκκινο πλαίσιο γύρω από τον διακόπτη 1. Αυτή η ένδειξη υπάρχει ώστε να φαίνεται, ποιόν ταλαντωτή αφορούν οι ρυθμίσεις που γίνονται εντός του πλαισίου ρυθμίσεων ταλαντωτών. Ξεκινώντας η εφαρμογή του AudioSynth, ο διακόπτης 1 είναι αυτός που περικλείεται από το κόκκινο πλαίσιο (default κατάσταση). Για να επέμβουμε στις ρυθμίσεις κάποιου άλλου ταλαντωτή,

χρειάζεται να μεταφέρουμε το κόκκινο πλαίσιο σε κάποιον από τους άλλους διακόπτες του grid. Αυτό γίνεται κάνοντας δεξί κλικ πάνω στον αντίστοιχο διακόπτη.

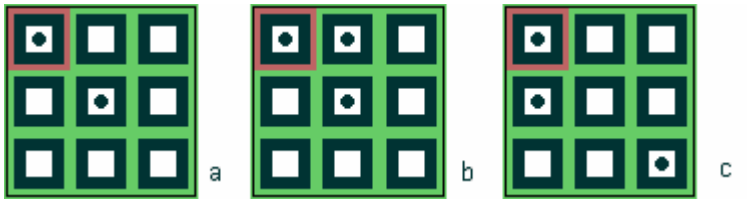
## 2.2 Κυκλώματα Διαμόρφωσης Πλάτους

Εξετάζουμε καταρχήν την περίπτωση όπου και τα δύο είδη διαμόρφωσης είναι τύπου AM:

Εικόνα 2.3

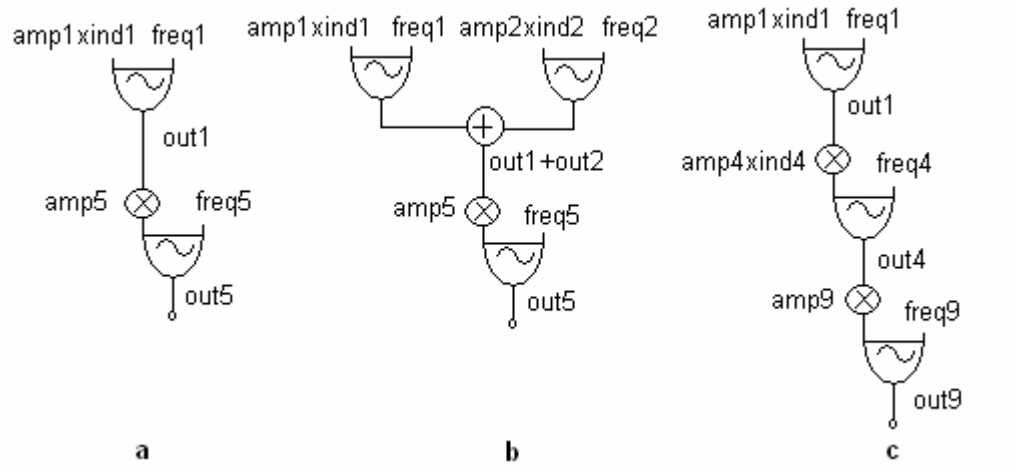
Ταυτόχρονα, ταλαντωτές δύο τουλάχιστον διαφορετικών γραμμών έχουν τεθεί σε κατάσταση ON. Σε αυτήν την περίπτωση το άθροισμα των σημάτων των ταλαντωτών που αντιστοιχούν στις υψηλότερες οριζόντιες γραμμές του grid διαμορφώνει τους ταλαντωτές της αμέσως επόμενης γραμμής της οποίας τουλάχιστον ένας διακόπτης είναι σε κατάσταση ON.

Ενδεικτικά θα εξηγήσουμε τις τρεις παρακάτω πιθανές καταστάσεις του grid:



Εικόνα 2.4

Τα block διαγράμματα που αντιστοιχούν στις καταστάσεις a, b και c του πλέγματος ταλαντωτών, έχουν αντίστοιχα ως εξής:



Εικόνα 2.5

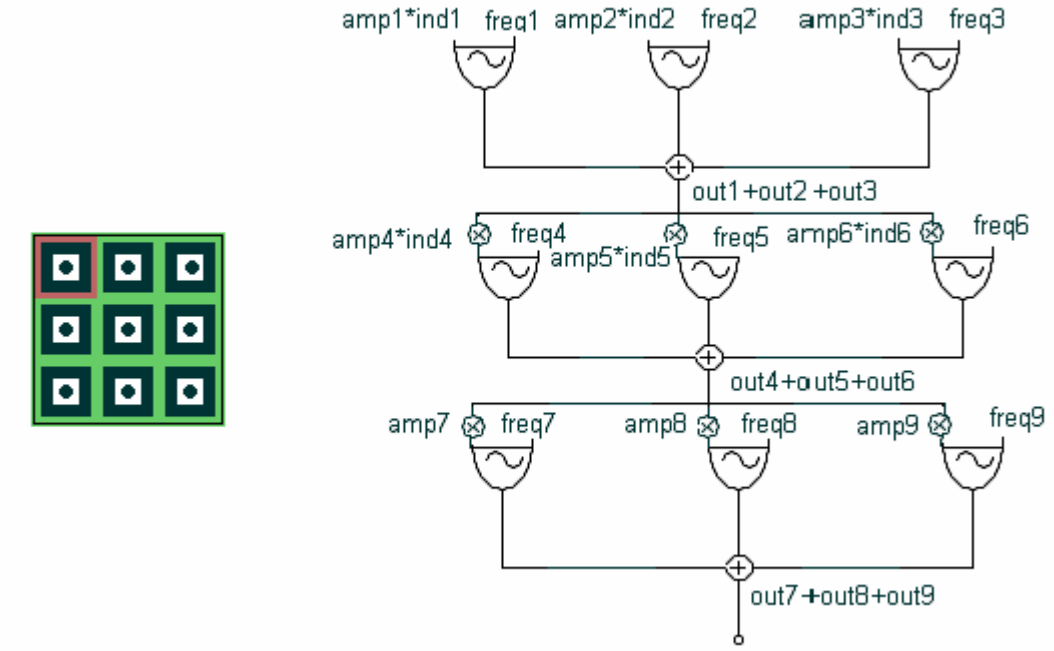
Στο διάγραμμα a έχουμε την πιο απλή περίπτωση διαμόρφωσης πλάτους όπου ένας ταλαντωτής είναι διαμορφωτής ενός άλλου. Παρατηρούμε ότι δεν έχουμε να

κάνουμε με την κλασική διαμόρφωση πλάτους στην οποία η έξοδος του διαμορφωτή θα προσθετόταν στο πλάτος του ταλαντωτή–φορέα. Αντί αυτού, η έξοδος του διαμορφωτή πολλαπλασιάζεται με το πλάτος του ταλαντωτή–φορέα. Το ηχητικό αποτέλεσμα αυτού του τρόπου διαμόρφωσης είναι παρόμοιο με αυτό της κλασικής διαμόρφωσης πλάτους και οι λόγοι επιλογής του θα εξηγηθούν παρακάτω. Παρατηρούμε επίσης ότι το πλάτος του διαμορφωτή πολλαπλασιάζεται με έναν συντελεστή ind1 που αποτελεί τον συντελεστή διαμόρφωσης (modulation index). Ο συντελεστής αυτός όπως εξετάστηκε και στο προηγούμενο κεφάλαιο μπορεί να πάρει τιμές από 0 ως 1.

Στο διάγραμμα b έχουμε την περίπτωση διαμόρφωσης με πολλαπλούς ταλαντωτές διαμόρφωσης εν παραλλήλω. Σε αυτήν την περίπτωση το σήμα διαμόρφωσης δεν είναι η έξοδος ενός μόνο ταλαντωτή διαμόρφωσης, αλλά το άθροισμα των εξόδων των ταλαντωτών διαμόρφωσης που στην συγκεκριμένη περίπτωση είναι οι ταλαντωτές 1 και 2. Το σήμα αυτό πολλαπλασιάζεται και πάλι με το πλάτος του ταλαντωτή–φορέα, ενώ όπως και στην προηγούμενη περίπτωση, ο κάθε διαμορφωτής συμβάλλει στην ολική διαμόρφωση σύμφωνα με την τιμή του συντελεστή διαμόρφωσης που του αντιστοιχεί.

Στο διάγραμμα c έχουμε δύο διαμορφωτές εν σειρά. Με τον ίδιο τρόπο όπως και πριν, ο ταλαντωτής 1 διαμορφώνει το πλάτος του ταλαντωτή 4 της επόμενης οριζόντιας γραμμής, ο οποίος με τη σειρά του διαμορφώνει τον ταλαντωτή 9 της τρίτης και τελευταίας γραμμής του πλέγματος.

Στην εικόνα 2.6 βλέπουμε την περίπτωση όπου όλοι οι διακόπτες του πλέγματος ταλαντωτών είναι σε κατάσταση ON, οπότε το διάγραμμα του κυκλώματος έχει ως εξής:



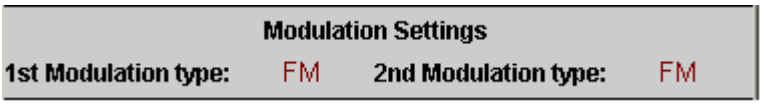

Εικόνα 2.6

Όπως φαίνεται από το παραπάνω block διάγραμμα στην περίπτωση που όλοι οι διακόπτες του πλέγματος ταλαντωτών είναι σε κατάσταση ON, υλοποιούνται όλα τα

είδη διαμόρφωσης πλάτους που εξετάσαμε παραπάνω. Έχουμε δηλαδή ταυτόχρονα διαμόρφωση πλάτους με χρήση πολλαπλών ταλαντωτών διαμόρφωσης και εν σειρά και παράλληλα καθώς και με χρήση πολλαπλών ταλαντωτών – φορέων (ταλαντωτές 7,8 και 9).

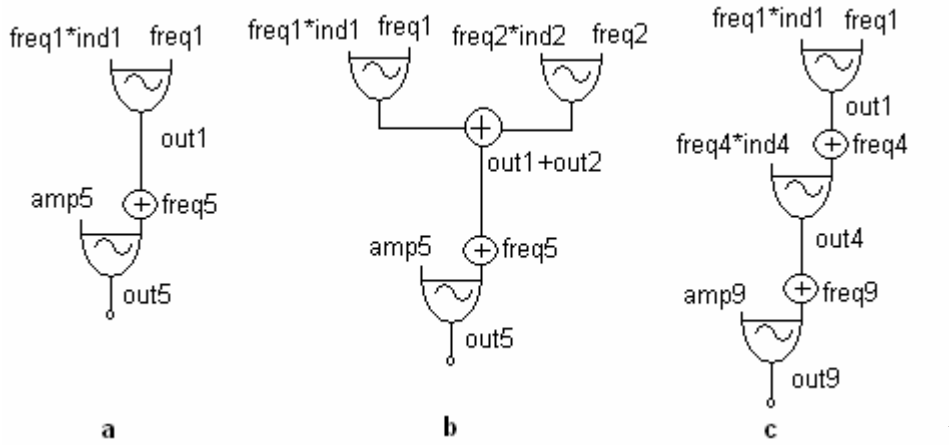
## 2.3 Κυκλώματα Διαμόρφωσης Συχνότητας

Για να έχουμε πολλαπλή διαμόρφωση συχνότητας (Multiple FM) το πλαίσιο επιλογής τύπων διαμόρφωσης θα έχει την παρακάτω μορφή:



Εικόνα 2.7

Θα εξετάσουμε αυτόν τον τύπο σύνθετης διαμόρφωσης χρησιμοποιώντας τα κυκλώματα a, b και c που χρησιμοποιήσαμε και για την εξέταση της διαμόρφωσης πλάτους (εικόνα 2.4). Παρακάτω, βλέπουμε τα block διαγράμματα που αντιστοιχούν σε αυτές τις τρεις περιπτώσεις:



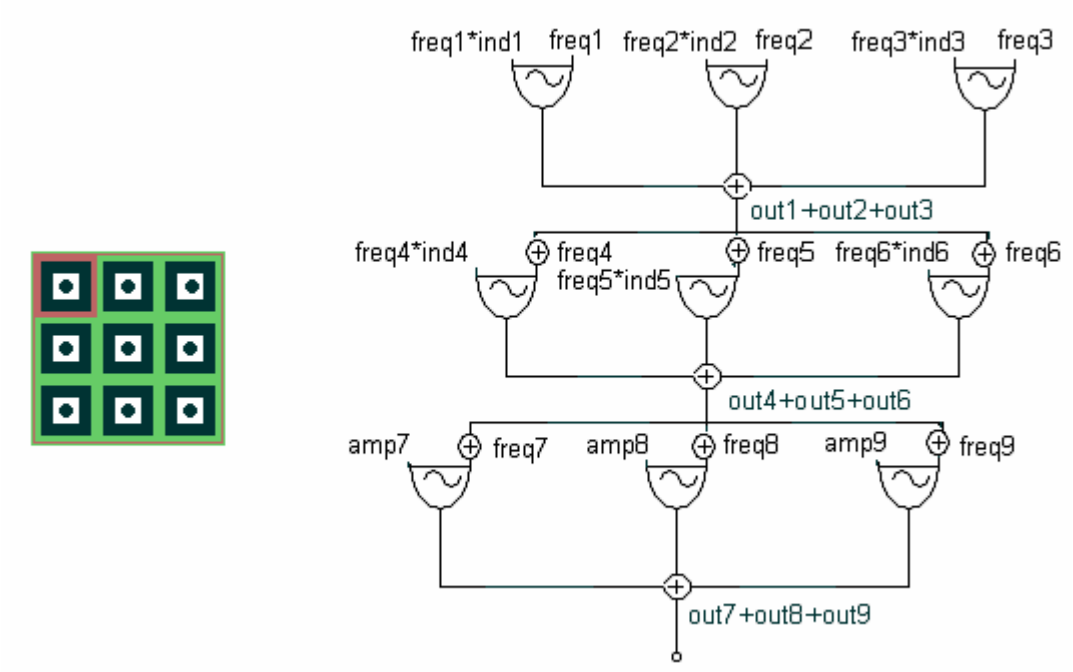
Εικόνα 2.8

Στο διάγραμμα a της εικόνας 2.8 παριστάνεται η πιο απλή περίπτωση διαμόρφωσης συχνότητας που μπορεί να πραγματοποιήσει ο χρήστης του AudioSynth. Πρόκειται για την κλασική διαμόρφωση συχνότητας, όπου η έξοδος ενός ημιτονικού ταλαντωτή (διαμορφωτής) προστίθεται στη συχνότητα ενός άλλου ημιτονικού ταλαντωτή (φορέας). Παρατηρούμε ότι το πλάτος του διαμορφωτή ισοδυναμεί με τη συχνότητα του πολλαπλασιασμένη με ένα συντελεστή διαμόρφωσης. Ο συντελεστής της FM διαμόρφωσης στο συγκεκριμένο περιβάλλον σύνθεσης, παίρνει τιμές από 0 ως 20.

Στο διάγραμμα b έχουμε την περίπτωση της διαμόρφωσης συχνότητας με πολλαπλούς διαμορφωτές εν παραλλήλω. Η συχνότητα του ταλαντωτή φορέα διαμορφώνεται από το άθροισμα των εξόδων των δύο διαμορφωτών.

Στο διάγραμμα c έχουμε την περίπτωση της FM με δύο διαμορφωτές εν σειρά. Σε αυτήν την περίπτωση, όπως φαίνεται και στο διάγραμμα, ένας ταλαντωτής διαμορφώνει τα συχνότητα ενός ταλαντωτή που διαμορφώνει τη συχνότητα ενός τρίτου ταλαντωτή (φορέας).

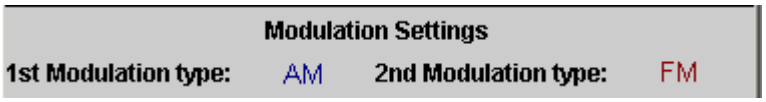
Είναι κατανοητό ότι μπορεί κανείς θέτοντας σε κατάσταση ON τους κατάλληλους διακόπτες στο grid να επιτύχει και διαμόρφωση πολλαπλών ταλαντωτών-φορέων. Τέλος όλες οι παραπάνω περιπτώσεις FM διαμόρφωσης μπορεί να συνυπάρξουν όπως φαίνεται και από το παρακάτω διάγραμμα που αντιστοιχεί στην κατάσταση του πλέγματος ταλαντωτών στην οποία όλοι οι ταλαντωτές είναι σε κατάσταση ON:




Εικόνα 2.9

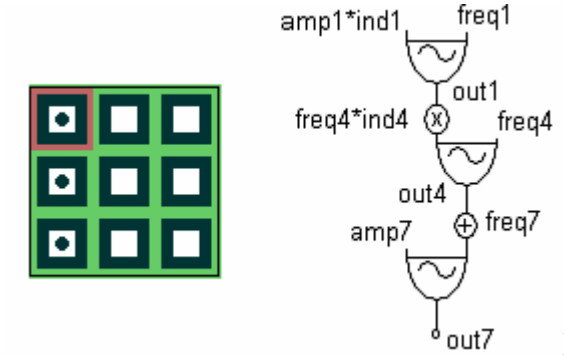
## 2.4 AM FM

Μέχρι τώρα εξετάσαμε τις περιπτώσεις όπου και τα δύο επίπεδα διαμορφώσεων του AudioSynth ήταν του ίδιου τύπου. Αρκετά ενδιαφέρουσες ηχητικά όμως είναι και οι περιπτώσεις όπου οι δύο διαμορφώσεις είναι διαφορετικού τύπου. Αυτές τις περιπτώσεις θα εξετάσουμε τώρα, αρχίζοντας από την περίπτωση στην οποία το πλαίσιο επιλογής τύπων διαμόρφωσης έχει την παρακάτω μορφή.




Εικόνα 2.10

Το πιο απλό κύκλωμα στην κατηγορία αυτού του τύπου διαμόρφωσης είναι η περίπτωση στην οποία έχουμε έναν μόνο διακόπτη σε κατάσταση ON, σε κάθε οριζόντια γραμμή του πλέγματος ταλαντωτών (Εικόνα 2.11):



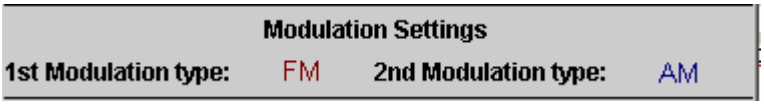
Εικόνα 2.11

Στο διάγραμμα της εικόνας 2.11 βλέπουμε ότι ο ταλαντωτής 1 διαμορφώνει το πλάτος του ταλαντωτή 4, η έξοδος του οποίου διαμορφώνει τελικά τη συχνότητα του ταλαντωτή 7.

Σε αυτό το σημείο μπορεί να εξηγηθεί γιατί επιλέχθηκε η διαμόρφωση πλάτους να γίνεται με πολλαπλασιασμό ενός σήματος με το πλάτος ενός ταλαντωτή φορέα, αντί να γίνεται με πρόσθεση όπως συμβαίνει στην κλασική διαμόρφωση πλάτους στη σύνθεση ήχου. Αν είχαμε κλασική διαμόρφωση πλάτους το πλάτος του ταλαντωτή 4 θα ήταν ίσο με (freq4*ind4 +out1). Η τιμή του freq4 μπορεί να πάρει τιμές έως και 10000 (10kHz) . Η έξοδος του ταλαντωτή 1 παίρνει τιμές από 0 ως 1. Έτσι στην περίπτωση που η συχνότητα του ταλαντωτή είναι πάνω από μερικές δεκάδες Hz η AM διαμόρφωση στον ταλαντωτή 4 θα τον έκανε να διαμορφώνει τον ταλαντωτή 7 με 1Hz μέγιστη μεταβαλλόμενη απόκλιση. Αυτή η απόκλιση βέβαια δεν είναι δυνατό να γίνει ακουστή. Αντίθετα στην περίπτωση όπου η έξοδος του ταλαντωτή 1 πολλαπλασιάζεται με τη συχνότητα και το συντελεστή διαμόρφωσης ind4, η συχνότητα του ταλαντωτή-φορέα 7 διαμορφώνεται ουσιαστικά από έναν ημιτονικά μεταβαλλόμενο συντελεστή διαμόρφωσης. Έτσι θα μπορούσαμε να πούμε ότι το διάγραμμα της εικόνας 1.14 περιγράφει διαμόρφωση συχνότητας με περιοδικά μεταβαλλόμενο συντελεστή διαμόρφωσης ο οποίος παίρνει τιμές από 0 ως ind4. Το ηχητικό αποτέλεσμα μπορεί να είναι αρκετά εντυπωσιακό.

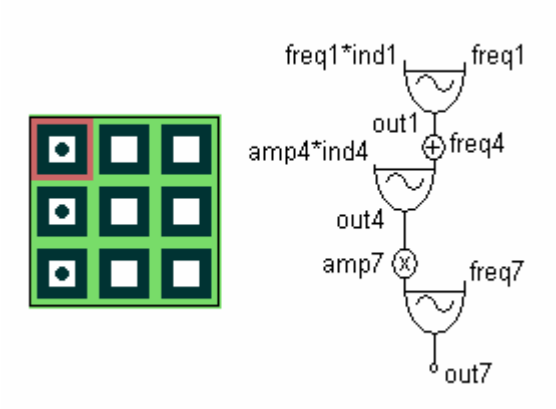
## 2.5 FM AM

Ο τελευταίος πιθανός συνδυασμός τύπων διαμόρφωσης, αφορά την παρακάτω ρύθμιση:



Εικόνα 2.12

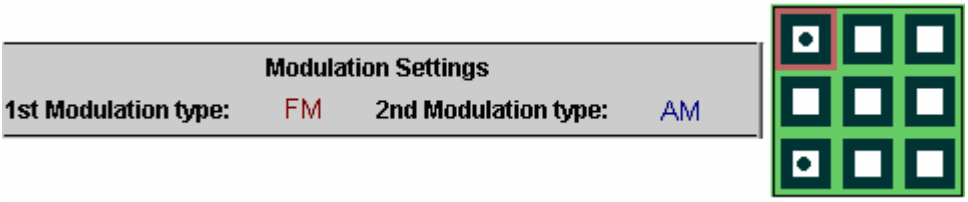
Θα εξετάσουμε την περίπτωση αυτή χρησιμοποιώντας το ίδιο κύκλωμα του πλέγματος ταλαντωτών, που χρησιμοποιήσαμε και για την περίπτωση της AM FM. Στην εικόνα 2.13 φαίνεται αυτή η κατάσταση του grid καθώς και το αντίστοιχο block διάγραμμα.




Εικόνα 2.13

Όπως βλέπουμε στο διάγραμμα της εικόνας 1.16, ο ταλαντωτής 1 διαμορφώνει τη συχνότητα του ταλαντωτή 4, ο οποίος με τη σειρά του διαμορφώνει το πλάτος του ταλαντωτή 7. Το αποτέλεσμα μιας τέτοιας υλοποίησης είναι η διαμόρφωση πλάτους του ταλαντωτή 7 από έναν διαμορφωτή (ταλαντωτής 4) ο οποίος έχει περιοδικά μεταβαλλόμενη συχνότητα. Θα μπορούσαμε να χαρακτηρίσουμε αυτήν την τεχνική σύνθεσης ήχου ως διαμόρφωση πλάτους με μεταβλητή συχνότητα διαμόρφωσης (modulation rate). Το ηχητικό αποτέλεσμα από μια τέτοια υλοποίηση μπορεί να είναι επίσης αρκετά εντυπωσιακό.

Τέλος, μία περίπτωση κυκλώματος όπου δεν είναι ιδιαίτερα εμφανής ως προς το αποτέλεσμά της, αφορά την παρακάτω περίπτωση:




Εικόνα 2.14

Σε αυτήν την περίπτωση ο χρήστης ίσως να αναρωτηθεί αν ο ταλαντωτής 1 θα διαμορφώνει το πλάτος ή την συχνότητα του ταλαντωτή 7. Η απάντηση είναι ότι θα έχουμε διαμόρφωση συχνότητας και όχι πλάτους. Ο τύπος της πρώτης διαμόρφωσης αφορά στο τι είδους διαμόρφωση θα κάνουν οι ταλαντωτές της πρώτης σειράς στους ταλαντωτές της δεύτερης ή αν αυτοί είναι απενεργοποιημένοι, σε αυτούς της τρίτης σειράς. Ο καθορισμός του τύπου της δεύτερης διαμόρφωσης αφορά στο τι είδους διαμόρφωση θα κάνουν οι ταλαντωτές της δεύτερης σειράς σε αυτούς της τρίτης. Στην συγκεκριμένη περίπτωση οι ταλαντωτές της δεύτερης σειράς είναι απενεργοποιημένοι οπότε αυτή η ρύθμιση δεν έχει κανένα νόημα. Συνεχίζει ωστόσο να παρέχεται η δυνατότητα αυτής της ρύθμισης στο χρήστη ώστε να έχει το επιθυμητό αποτέλεσμα σε περίπτωση που κάποια στιγμή ενεργοποιήσει κάποιον από τους ταλαντωτές 4, 5 και 6.

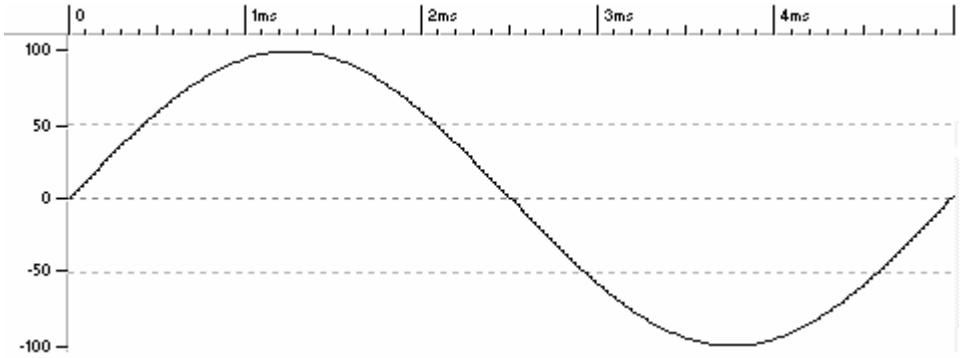
# 3. Ανάλυση Αποτελεσμάτων

Στο κεφάλαιο αυτό εξετάζεται η αξιοπιστία του AudioSynth, δηλαδή το κατά πόσο οι ρυθμίσεις του χρήστη στο interface της εφαρμογής ανταποκρίνονται στο παραγόμενο ηχητικό αποτέλεσμα. Για τον σκοπό αυτόν ηχογραφήθηκαν κάποια ηχητικά δείγματα της εξόδου του οργάνου, τα οποία αντιστοιχούν σε διαφορετικές ενδεικτικές καταστάσεις του. Σε κάθε μία περίπτωση, το σήμα που παράγεται στην έξοδο αναλύεται χρονικά και φασματικά με τη χρήση εξειδικευμένου λογισμικού. Στη συνέχεια οι μετρήσεις αυτές αντιπαραβάλλονται με το θεωρητικά αναμενόμενο αποτέλεσμα.

## 3.1 Απλή Ημιτονοειδής

Το πρώτο πράγμα που πρέπει να εξεταστεί είναι η έξοδος ημιτονοειδούς ταλάντωσης. Για τον σκοπό αυτό έχουν ηχογραφηθεί οι έξοδοι ενός ταλαντωτή για διαφορετικές ρυθμίσεις.

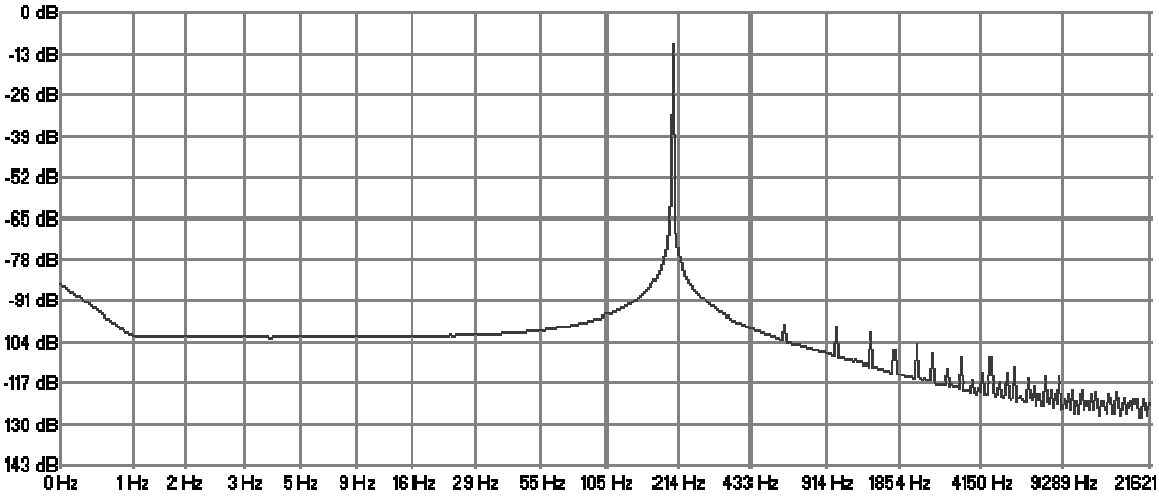
Οι πρώτες ρυθμίσεις αφορούν σε έναν ταλαντωτή τον οποίο έχουμε συντονίσει στη συχνότητα των 200 Hz. Στην απεικόνιση της κυματομορφής αυτού του ταλαντωτή είναι αναμενόμενο να δούμε μια ημιτονική κυματομορφή με περίοδο 1/200 sec = 5 ms. Η απεικόνιση της εξόδου του ταλαντωτή για μία περίοδο αυτού φαίνεται παρακάτω:




Εικόνα 3.1

Πράγματι, στην παραπάνω εικόνα βλέπουμε μια ημιτονοειδή κυματομορφή με περίοδο που δείχνει να είναι πολύ κοντά στα 5 ms και επομένως η συχνότητα θα είναι 200 Hz.

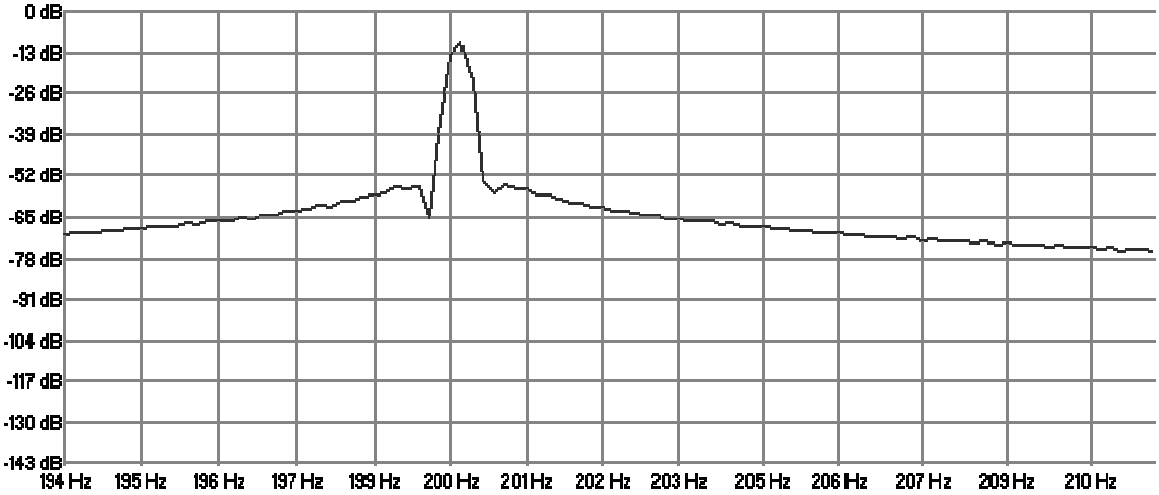
Στην εικόνα 3.2 βλέπουμε το φάσμα εξόδου του ταλαντωτή. Η ανάλυση αυτή γίνεται με χρήση FFT (Fast Fourier Transformation), με παράθυρο ανάλυσης Hamming, μέγεθος παραθύρου 262144 και επικάλυψη 25%. Για συχνότητα δειγματοληψίας 44100 δείγματα ανά δευτερόλεπτο, που είναι και η συχνότητα δειγματοληψίας των δειγμάτων μας, η ακρίβεια που μπορεί να προκύψει από την ανάλυση της παρακάτω εικόνας είναι της τάξης των 0.2 Hz. Αυτόν τον τύπο FFT θα χρησιμοποιήσουμε σε όλες τις αναλύσεις φάσματος αυτού του κεφαλαίου.



Εικόνα 3.2

Στο παραπάνω διάγραμμα φαίνεται ότι πράγματι κοντά στα 200Hz υπάρχει φασματική έξαρση, αλλά παρατηρούμε και την ύπαρξη άλλων συχνοτήτων. Οι συχνότητες αυτές αποτελούν θόρυβο ο οποίος είτε παράγεται από τον υπολογισμό του βήματος διαβάσματος του ταλαντωτή, είτε οφείλεται στη διαδικασία της ηχογράφησης στον υπολογιστή.

Από την παραπάνω εικόνα δεν μπορούμε να διακρίνουμε αν πράγματι η φασματική έξαρση βρίσκεται ακριβώς στα 200 Hz. Σε αυτό θα μας βοηθήσει η παρακάτω εικόνα στην οποία φαίνεται η παραπάνω ανάλυση για μια πολύ μικρή περιοχή συχνοτήτων γύρω από τα 200 Hz.



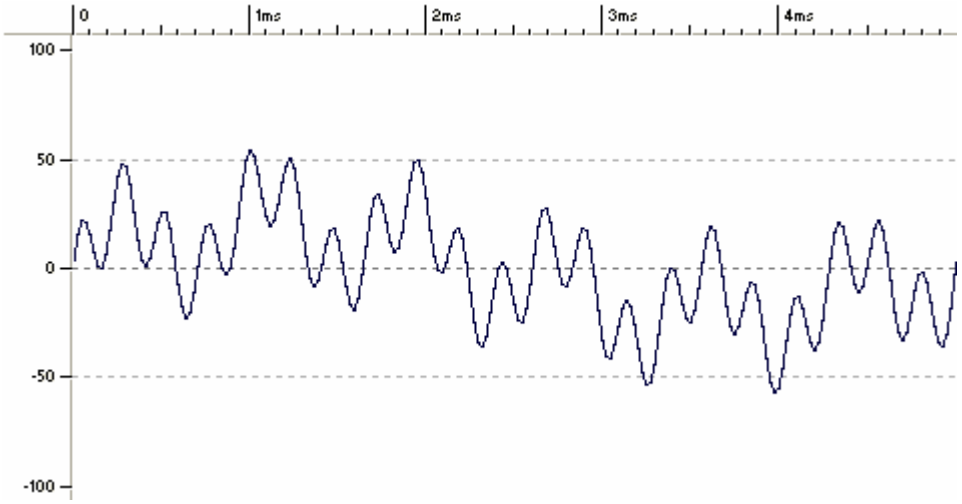
Εικόνα 3.3

Από το παραπάνω διάγραμμα βλέπουμε ότι η συχνότητα είναι περίπου στα 200,2 Hz δηλαδή πολύ κοντά στα 200Hz και με μια απόκλιση της τάξης των 0,2 Hz που όπως έχει ήδη αναφερθεί, είναι κοντά στο σφάλμα που μπορεί να προκύψει από τον τύπο της ανάλυσης που έχει επιλεγεί.

## 3.2 Πρόσθεση Ημιτονοειδών

Σε αυτό το σημείο θα εξεταστεί η έξοδος του AudioSynth για την περίπτωση που αυτή προκύπτει από την πρόσθεση ταλαντωτών. Συγκεκριμένα θα εξεταστεί το σήμα που προκύπτει από την πρόσθεση των εξόδων τριών ταλαντωτών που έχουν το ίδιο πλάτος και συχνότητες 200 , 1200 και 4200 Hz.

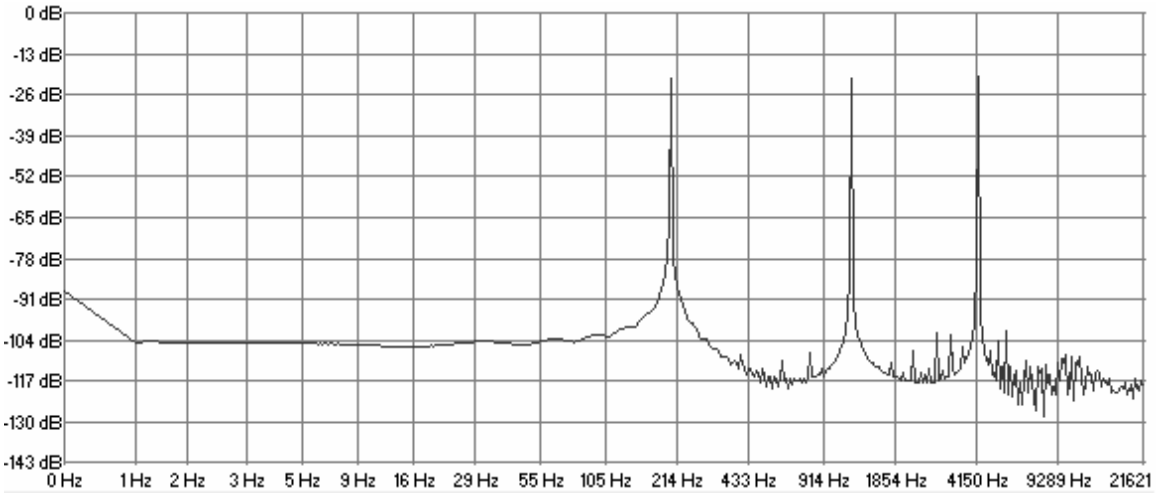
Από την πρόσθεση των τριών αυτών σημάτων περιμένουμε να προκύψει μια σύνθετη περιοδική κυματομορφή με περίοδο ίση με αυτή του ταλαντωτή με τη χαμηλότερη συχνότητα, δηλαδή του ταλαντωτή ο οποίος έχει συντονιστεί στα 200 Hz. Στην παρακάτω εικόνα φαίνεται η απεικόνιση μιας περιόδου της κυματομορφής.




Εικόνα 3.4

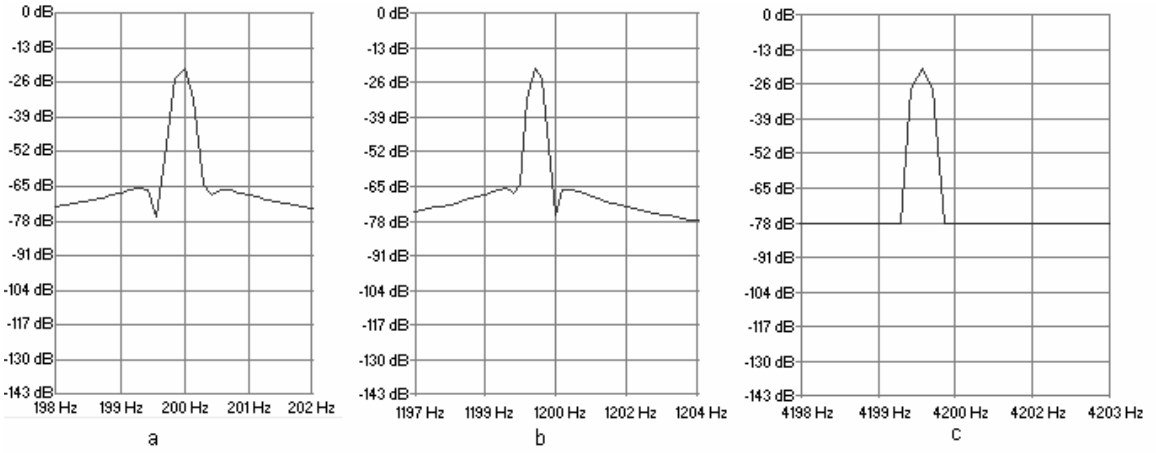
Όπως βλέπουμε η περίοδος της παραπάνω σύνθετης κυματομορφής είναι πράγματι 5 ms, χρόνος που αντιστοιχεί στη συχνότητα των 200 Hz.

Το συχνοτικό φάσμα στην συγκεκριμένη περίπτωση περιμένουμε φυσικά να περιλαμβάνει τρεις συχνοτικές εξάρσεις στα σημεία των 200, 1200 και 4200 Hz. Η ανάλυση αυτού του φάσματος φαίνεται παρακάτω:




Εικόνα 3.5

Στο διάγραμμα της εικόνας 3.5 βλέπουμε ότι πράγματι υπάρχουν οι φασματικές εξάρσεις που ήταν και θεωρητικά αναμενόμενες. Βλέπουμε επίσης ότι τα πλάτη τους είναι πράγματι τα ίδια. Φυσικά οι ακριβείς συχνότητες στις οποίες αντιστοιχούν αυτές οι εξάρσεις δεν είναι ορατές. Για αυτό το λόγο στην παρακάτω εικόνα βλέπουμε τρία διαγράμματα που αποτελούν μεγεθύνσεις στις τρεις συχνοτικές περιοχές του παραπάνω διαγράμματος.




Εικόνα 3.6

Όπως βλέπουμε στις παραπάνω αναλύσεις φάσματος, οι αποκλίσεις από τις θεωρητικά αναμενόμενες τιμές είναι αρκετά μικρές και κάτω από τα 0,5 Hz.

## 3.3 Διαμόρφωση Πλάτους

Σε αυτό το σημείο θα εξετάσουμε τη διαμόρφωση πλάτους ενός ταλαντωτή από κάποιον άλλο. Όπως έχει ήδη ειπωθεί, στην περίπτωση της διαμόρφωσης πλάτους ενός ταλαντωτή, το πλάτος του ταλαντωτή-φορέα πολλαπλασιάζεται με την έξοδο του διαμορφωτή. Δηλαδή ισχύει:

$$S(t) = A_c *(A_m *\sin(2\pi f_m t)) *\sin(2\pi f_c t) = A_c *A_m *\sin(2\pi f_m t) *\sin(2\pi f_c t)$$
$$= ½ A_c A_m *[\sin(2\pi[fc-fm]t) + \sin(2\pi[fc+fm]t)]$$

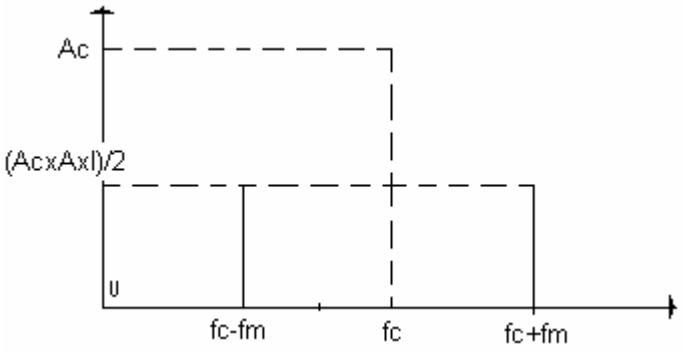
όπου:

$A_c$, $A_m$ το πλάτος του ταλαντωτή-φορέα και του διαμορφωτή αντίστοιχα
$f_c$, $f_m$ η συχνότητα φορέα και η συχνότητα διαμόρφωσης αντίστοιχα

Το πλάτος $A_m$ του ταλαντωτή διαμόρφωσης θα ισούται με A*I, όπου A το πλάτος που δίνεται από το ανάλογο slider και I ο συντελεστής διαμόρφωσης. Αντικαθιστώντας στην παραπάνω σχέση με A*I, θα ισχύει:

$$S(t) =½ A_c A*I [\sin(2\pi[fc-fm]t) + \sin(2\pi[fc+fm]t)]$$

Η παραπάνω σχέση αναπαριστά ένα σύνθετο σήμα που προκύπτει από τη πρόσθεση δύο ημιτονοειδών με συχνότητες fc-fm και fc+fm. Στην παρακάτω εικόνα βλέπουμε τη μορφή που αναμένεται να έχει το φάσμα εξόδου.

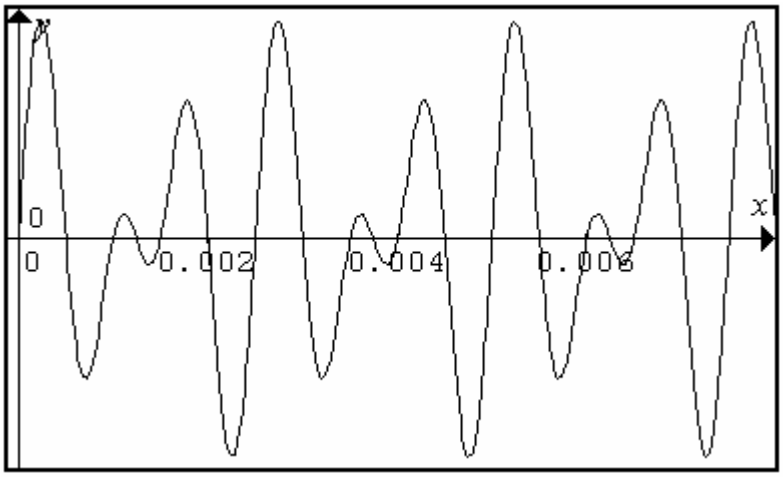


Εικόνα 3.7

Για την εξέταση της AM σύνθεσης στο AudioSynth θα χρησιμοποιήσουμε έναν φορέα με πλάτος 1 (0 dB), συχνότητα 1000Hz και έναν διαμορφωτή επίσης με πλάτος 1, συχνότητα 200 Hz και συντελεστή διαμόρφωσης 0,5. Αντικαθιστώντας στην προηγούμενη σχέση για το σήμα εξόδου θα ισχύει:

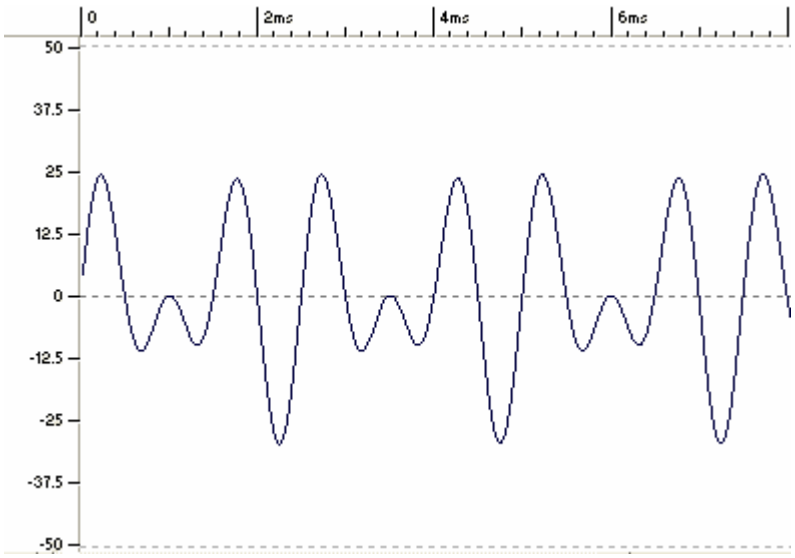
$$S(t)= 0.25 [\sin(2\pi 800t) +\sin(2\pi 1200t)]$$

Η κυματομορφή που προκύπτει από την παραπάνω σχέση αναμένεται να έχει την παρακάτω μορφή:



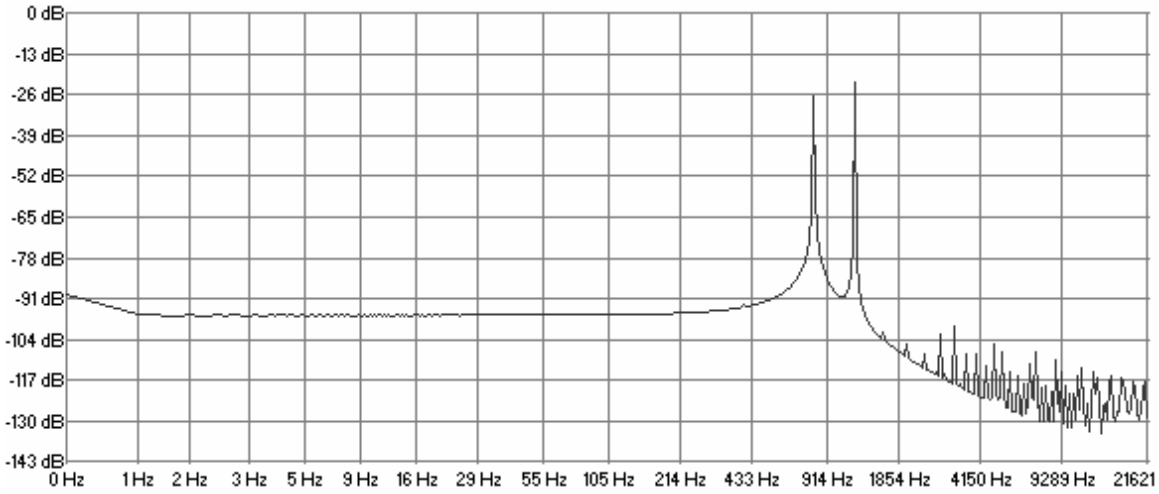

Εικόνα 3.8

Στην παρακάτω εικόνα φαίνεται η απεικόνιση της κυματομορφής όπως αυτή προέκυψε ύστερα από την ηχογράφηση της εξόδου του AudioSynth με τις παραπάνω ρυθμίσεις.



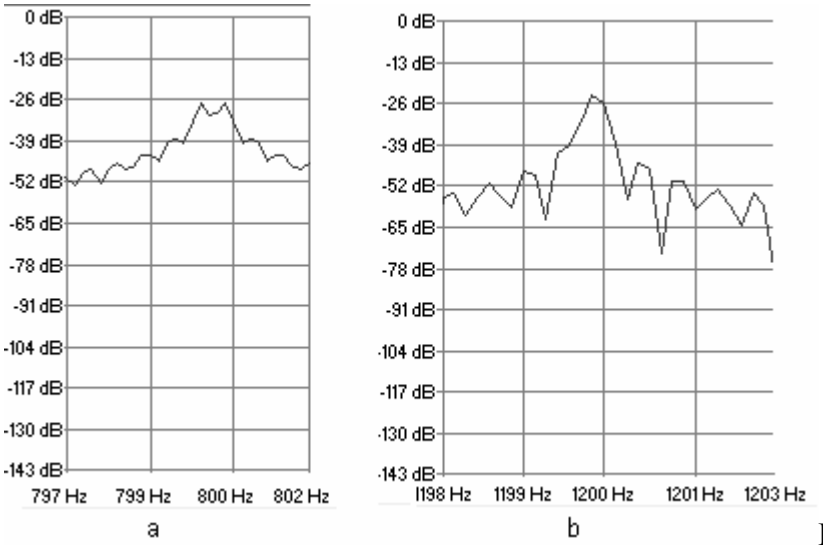

Εικόνα 3.9

Οι δύο κυματομορφές των προηγούμενων εικόνων παρουσιάζουν μία ομοιότητα χωρίς βέβαια να συμπίπτουν τελείως. Ο λόγος για αυτό ίσως να είναι η ύπαρξη θορύβου τον οποίο μπορούμε να δούμε και στο επόμενο διάγραμμα που αποτελεί την ανάλυση φάσματος αυτής της κυματομορφής.




Εικόνα 3.10

Στο παραπάνω φάσμα παρατηρούμε την ύπαρξη των δύο peak που περιμέναμε στις δύο πλευρικές συχνότητες των 800 και 1200 Hz. Το ότι είναι πράγματι αυτές οι συχνότητες, μπορούμε να το παρατηρήσουμε καλύτερα στα διαγράμματα της εικόνας 3.11. Αυτό που θεωρητικά δεν ήταν αναμενόμενο είναι η διαφορά πλάτους στις δύο πλευρικές αφού δεν προκύπτει κάτι τέτοιο από τον τύπο που εξετάσαμε προηγουμένως.




Εικόνα 3.11

Τα δύο παραπάνω διαγράμματα δείχνουν ότι και πάλι η απόκλιση από τις αναμενόμενες συχνότητες είναι αρκετά μικρή. Ωστόσο σε αυτήν την περίπτωση παρατηρούμε ότι αυτά τα peak δεν είναι τόσο απότομα όπως ήταν στην περίπτωση που προέκυπταν από την πρόσθεση ταλαντωτών. Μικρές περιοχές συχνοτήτων γύρω από τις δύο πλευρικές είναι επίσης φασματικά έντονες.

## 3.4 Διαμόρφωση Συχνότητας

Κατά τη διαμόρφωση συχνότητας ενός ταλαντωτή από έναν άλλο, στη συχνότητα του ταλαντωτή φορέα προστίθεται η έξοδος του διαμορφωτή. Για το σήμα εξόδου λοιπόν S(t) θα ισχύει:

$$S(t) = A_c \times \sin(2\pi fc + (I \times A_m \sin(2\pi f_m))$$

Όπου :

$A_c$, $A_m$ , το πλάτος του φορέα και του διαμορφωτή αντίστοιχα

$f_c$, $f_m$ , η συχνότητα φορέα και η συχνότητα διαμορφωτή αντίστοιχα και

I , ο συντελεστής διαμόρφωσης

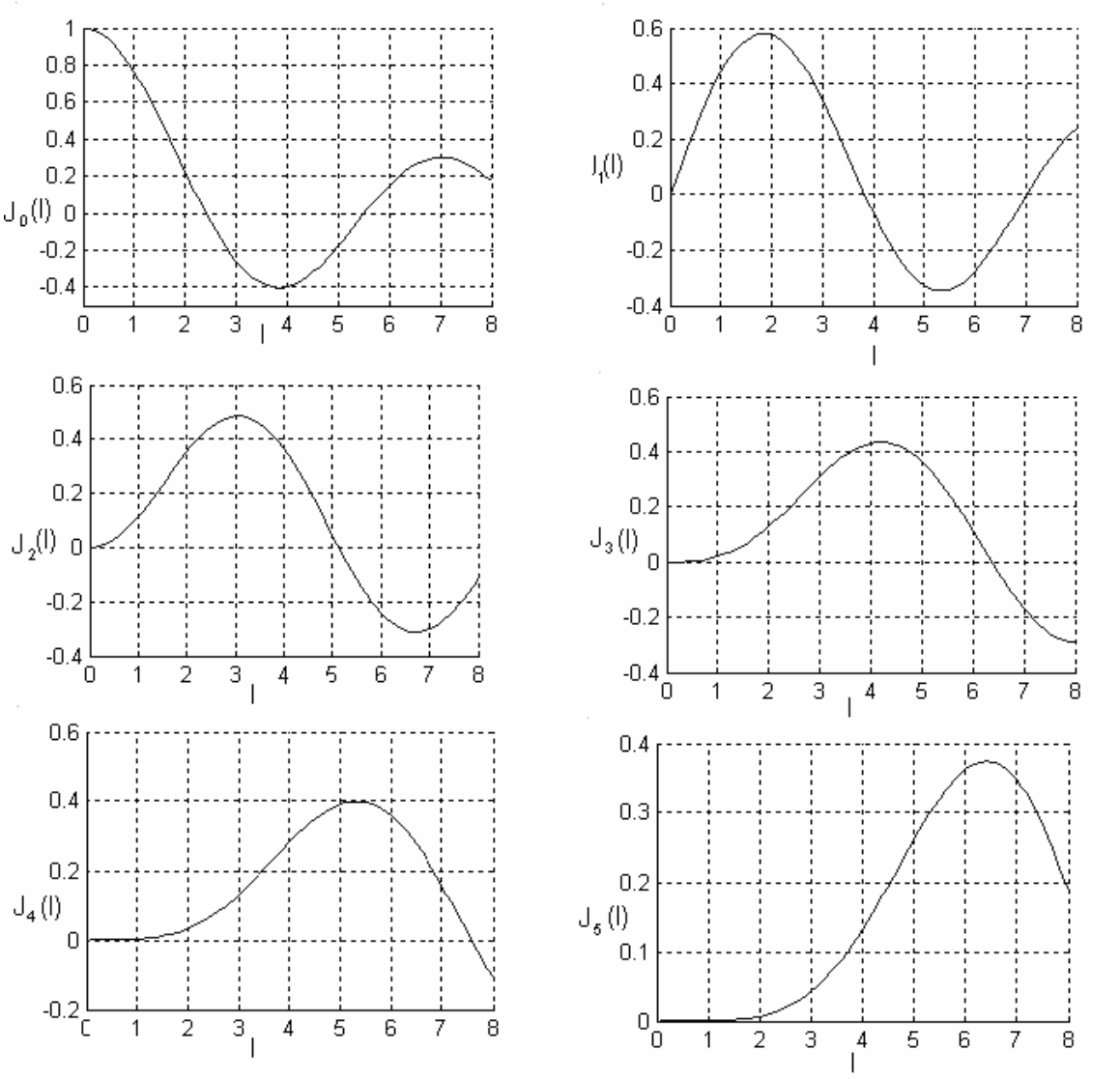
Η διαμόρφωση συχνότητας ενός ταλαντωτή από έναν άλλο, θεωρητικά οδηγεί σε ένα φάσμα το οποίο περιέχει τη συχνότητα φορέα και ένα πλήθος από παράπλευρες, οι οποίες απέχουν από τη συχνότητα φορέα κατά πολλαπλάσια της συχνότητας διαμόρφωσης. Το εύρος ζώνης (bandwidth) του παραγόμενου φάσματος μπορεί να διατυπωθεί ως:

$$\text{FM bandwidth} = 2 \times f_m \times (I+1)$$

Ο υπολογισμός του πλήθους των παράπλευρων συχνοτήτων και του πλάτους αυτών γίνεται με χρήση των συναρτήσεων Bessel πρώτου τύπου. Ο τύπος αυτών των συναρτήσεων φαίνεται παρακάτω:

$$J_k(I)=I^k \sum_{m=0}^{\infty} \frac{(-1)^m \times I^{2m}}{2^{2m+k}\, m!(k+m)!} \text{ , όπου } \in N$$

Στα διαγράμματα που ακολουθούν, φαίνονται αυτές οι συναρτήσεις για διάφορες τιμές του k:




Εικόνα 3.12

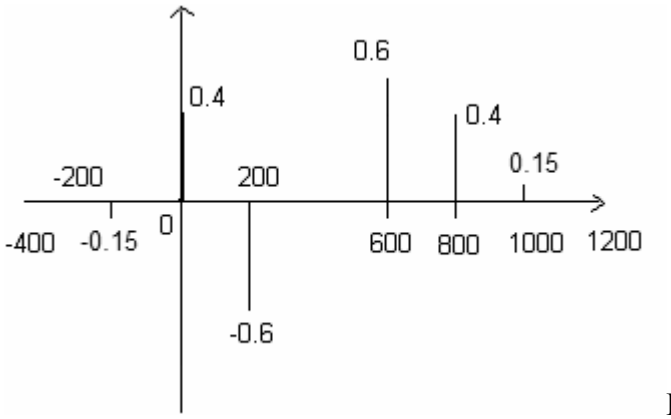
Έχοντας σαν δεδομένο έναν συντελεστή διαμόρφωσης I σε κάθε πίνακα (άξονας x), βρίσκουμε τη συγκεκριμένη τιμή της συνάρτησης (άξονας y). Οι συνιστώσες του παραγόμενου φάσματος θα έχουνε τα παρακάτω χαρακτηριστικά:

| **συνιστώσα** | **κάτω πλευρά** | | **άνω πλευρά** | |
|---|---|---|---|---|
| **k** | **συχνότητα** | **πλάτος** | **συχνότητα** | **πλάτος** |
| 1 | $f_c$-$f_m$ | -$J_1$(I) | $f_c$+$f_m$ | $J_1$(I) |
| 2 | $f_c$-2$f_m$ | $J_2$(I) | $f_c$+2$f_m$ | $J_2$(I) |
| 3 | $f_c$-3$f_m$ | -$J_3$(I) | $f_c$+3$f_m$ | $J_3$(I) |
| 4 | $f_c$-4$f_m$ | $J_4$(I) | $f_c$+4$f_m$ | $J_4$(I) |
| 5 | $f_c$-5$f_m$ | -$J_5$(I) | $f_c$+5$f_m$ | $J_5$(I) |

Αν στον παραπάνω πίνακα, θέσουμε I=2, $f_c$ = 400 και $f_m$ = 200 έχουμε:

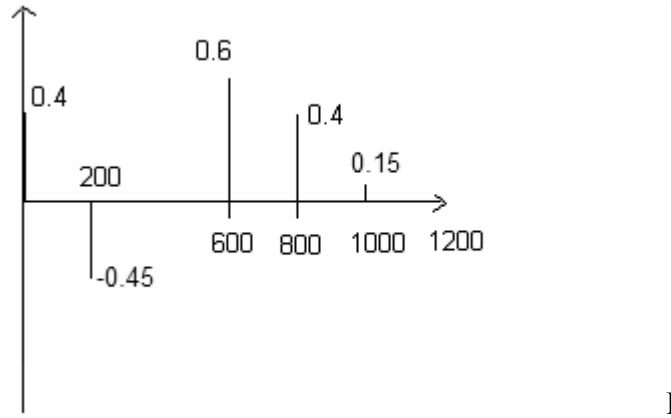
| συνιστώσα | κάτω πλευρά | | άνω πλευρά | |
|---|---|---|---|---|
| k | συχνότητα | πλάτος | συχνότητα | πλάτος |
| 1 | 200 | -0.6 | 600 | 0.6 |
| 2 | 0 | 0.4 | 800 | 0.4 |
| 3 | -200 | -0.15 | 1000 | 0.15 |
| 4 | -400 | 0 | 1200 | 0 |

Σύμφωνα με τις παραπάνω τιμές το φάσμα που θα προκύψει θα έχει τη μορφή της παρακάτω εικόνας.




Εικόνα 3.13

Οι αρνητικές τιμές συχνότητας αφορούν στην αντίστοιχη θετική τιμή συχνότητας με διαφορά φάσης $180^o$, άρα το φάσμα που θα προκύψει μετά την αναδίπλωση και τις σχετικές ακυρώσεις ή ενισχύσεις θα έχει τη μορφή της εικόνας 3.14.




Εικόνα 3.14

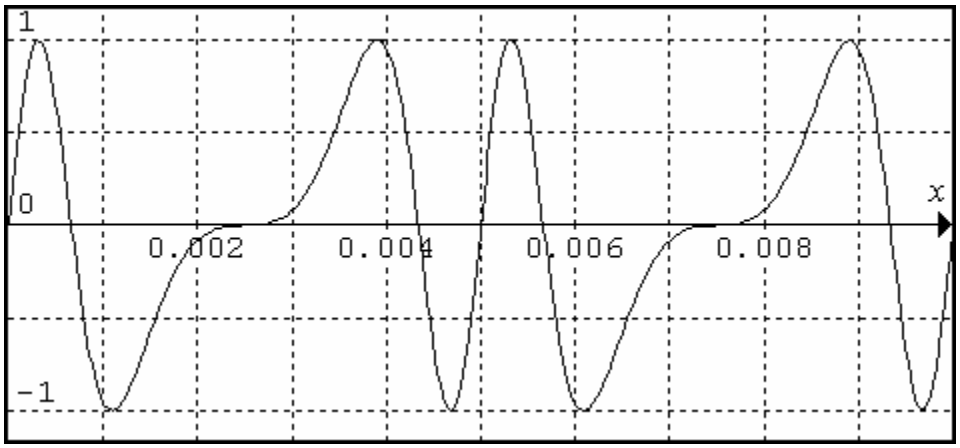
Παρατηρούμε ότι στα 200 Hz θα έχουμε μια συνιστώσα με πλάτος 0.5 και διαφορά φάσης $180^o$, καθώς και ότι θα έχουμε DC offset 0.4, αφού έχουμε πλάτος στα 0Hz. Είδαμε ότι το σήμα εξόδου περιγράφεται από την παρακάτω σχέση:

$$S(t) = A_c \text{ x } \sin(2\pi fct + (I \text{ x } A_m \sin(2\pi f_m t))$$

Αν σε αυτήν τη σχέση αντικαταστήσουμε τις μεταβλητές με τις τιμές που χρησιμοποιήσαμε πριν, καθώς και με την τιμή 1 για τα πλάτη, θα πάρει τη μορφή:

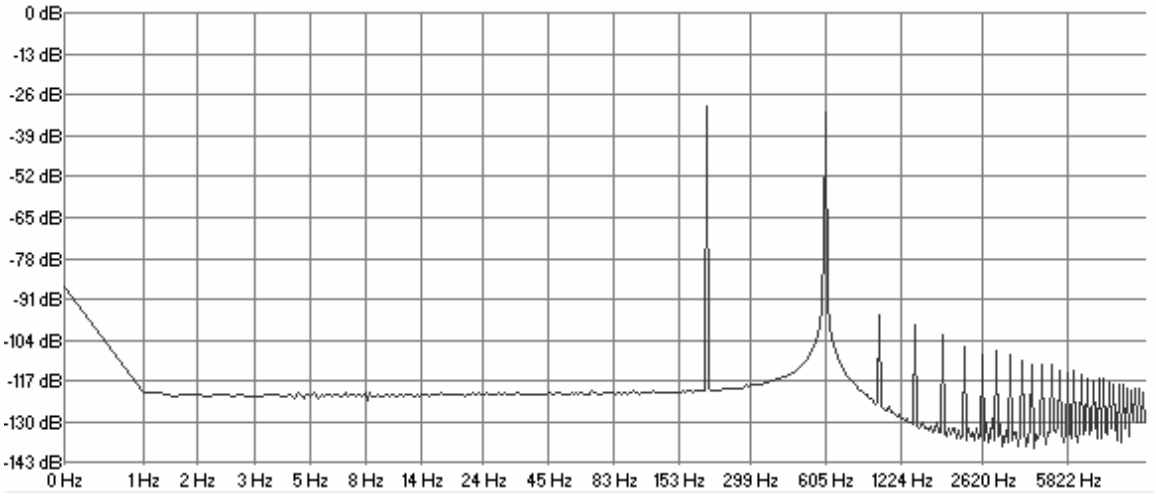
$$S(t) = \sin(2\pi 400t + (2 \sin(2\pi 200t))$$

Στο παρακάτω διάγραμμα βλέπουμε αυτήν ακριβώς τη συνάρτηση και επομένως τη μορφή που περιμένουμε να έχει η κυματομορφή στην έξοδο του AudioSynth.




Εικόνα 3.15

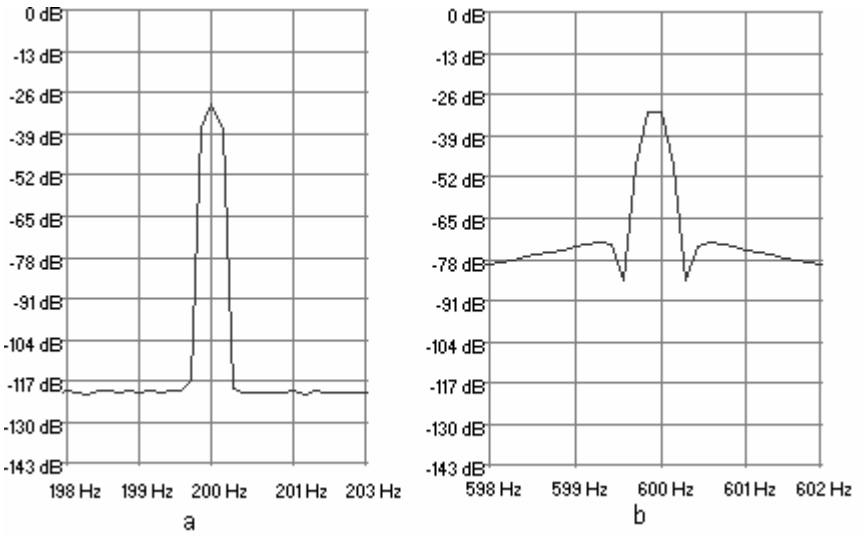
Στην παρακάτω εικόνα φαίνεται η απεικόνιση του φάσματος του AudioSynth για τις ρυθμίσεις που αναφέραμε προηγουμένως:




Εικόνα 3.16

Βλέπουμε ότι το φάσμα αποτελείται πράγματι από δύο υψηλά peak κοντά στις συχνότητες των 200 και 600 Hz όπως και στην εικόνα 3.14 όπου απεικονίζεται το θεωρητικά αναμενόμενο φάσμα. Επίσης εμφανίζονται και άλλες αρμονικές με απόσταση η μία από την άλλη πολλαπλάσιων της συχνότητας διαμόρφωσης. Ωστόσο το πλάτος αυτών είναι αισθητά μικρότερο.

Στην παρακάτω εικόνα βλέπουμε το παραπάνω διάγραμμα σε μεγέθυνση για τα δύο πρώτα και υψηλότερα peak.




Εικόνα 3.17

Παρατηρούμε ότι η συχνότητα αυτών των συνιστωσών είναι ακριβώς αυτή που ήταν και θεωρητικά αναμενόμενη. Επίσης τα πλάτη τους φαίνεται πως είναι ισοδύναμα, γεγονός που συμφωνεί με την ανάλυση. Αντίθετα, η ανάλυση δε συμφωνεί με την αναμενόμενη συχνότητα στα 800 Hz.

# 4. Συμπεράσματα

Σε σχέση με τους αλγόριθμους σύνθεσης ήχου που αναπτύχθηκαν στη συγκεκριμένη εργασία, σε γενικές γραμμές, διαπιστώθηκε συμφωνία μεταξύ του θεωρητικού μέρους των τεχνικών και των αποτελεσμάτων που προέκυψαν από τους αλγόριθμους. Κάποιες διαφοροποιήσεις μπορεί να προέρχονται είτε από εξωγενείς παράγοντες (θόρυβος κάρτας ήχου), είτε από τον ίδιο τον αλγόριθμο, κυρίως όσον αφορά τις μεθόδους υπολογισμού του βήματος διαβάσματος των πινάκων κυματομορφής. Ενδιαφέρον από ηχητικής πλευράς παρουσίασαν οι περιπτώσεις συνδυασμού διαφορετικών τύπων διαμόρφωσης. Παρατηρήθηκε επίσης, πως το ίδιο το γραφιστικό περιβάλλον προσδίδει μία ιδιαίτερη ηχοχρωματική προσέγγιση των τεχνικών αυτών.

Από την ενασχόληση με τον προγραμματισμό σε γλώσσα Java για τις ανάγκες της παρούσας εργασίας, προκύπτουν τα εξής συμπεράσματα:

1. Η γλώσσα Java είναι ιδανική γλώσσα για την γρήγορη ανάπτυξη γραφικών εξαιτίας των σχετικών βιβλιοθηκών που διαθέτει.

2. Η απλότητα της λογικής και κάποια ιδιαίτερα χαρακτηριστικά της γλώσσας όπως η απουσία δεικτών, διευκολύνουν σε μεγάλο βαθμό αρχάριους προγραμματιστές.

3. Η δημοτικότητα της γλώσσας έχει οδηγήσει στην ύπαρξη πολλών δικτυακών κοινοτήτων που ασχολούνται με την Java. Η πλούσια βιβλιογραφία γύρω από αυτή σε συνδυασμό με τα διάφορα εγχειρίδια που διαθέτει δωρεάν η Sun στις ιστοσελίδες της, αλλά και τις ομάδες συζήτησης (mailing lists), ο προγραμματιστής μπορεί να ενημερώνεται γρήγορα για τα χαρακτηριστικά της γλώσσας και τα bugs που μπορεί να υπάρχουν.

4. Το JavaSound API είναι αρκετά εύχρηστο και παρέχει στον προγραμματιστή τα απαραίτητα δομικά στοιχεία για την ανάπτυξη audio εφαρμογών. Ενδεικτικά αναφέρεται η δυνατότητα ανάγνωσης και δημιουργίας διαφόρων format ήχου και η εύκολη επικοινωνία με τις εγκατεστημένες κάρτες ήχου και το MIDI Hardware που είναι διαθέσιμο.

5. Η ανάπτυξη εφαρμογών σύνθεσης ήχου, διευκολύνεται από τον αντικειμενοστραφή χαρακτήρα της γλώσσας. Η ανάπτυξη αντικειμένων που αναπαριστούν δομικά στοιχεία της σύνθεσης ήχου και η ανάπτυξη αλγορίθμων για την συνεργασία αυτών ώστε να προκύψουν πιο σύνθετες εφαρμογές, αποτελεί μια πολύ καλή εκπαιδευτική άσκηση, ώστε ο προγραμματιστής να κατανοήσει καλύτερα τα ζητήματα της ψηφιακής σύνθεσης ήχου και της ψηφιακής ανάλυσης σήματος.

6. Το γεγονός ότι η Java είναι γλώσσα υψηλού επιπέδου, δεν σημαίνει ότι οι εφαρμογές ήχου που αναπτύσσονται με αυτή, θα είναι αργές κατά την εκτέλεση τους. Ακόμα όμως και αν προκύψει τέτοιο ζήτημα, είναι εφικτή η συνεργασία της Java με άλλες native γλώσσες προγραμματισμού (C, C++)

Σε μια περαιτέρω ανάπτυξη της εφαρμογής AudioSynth, θα μπορούσαν να γίνουν οι εξής προσθήκες:

1. Η δυνατότητα ανάγνωσης από τον χρήστη ενός αρχείου κειμένου βοήθειας (help), το οποίο θα περιέγραφε τα διάφορα κυκλώματα σύνθεσης ήχου που είναι δυνατόν να υλοποιηθούν.

2. Η δυνατότητα χειρισμού των διαφόρων ρυθμιστικών της εφαρμογής με τη χρήση MIDI controllers.

3. Η δυνατότητα αποθήκευσης και ανάκλησης των ρυθμίσεων του χρήστη (presets).

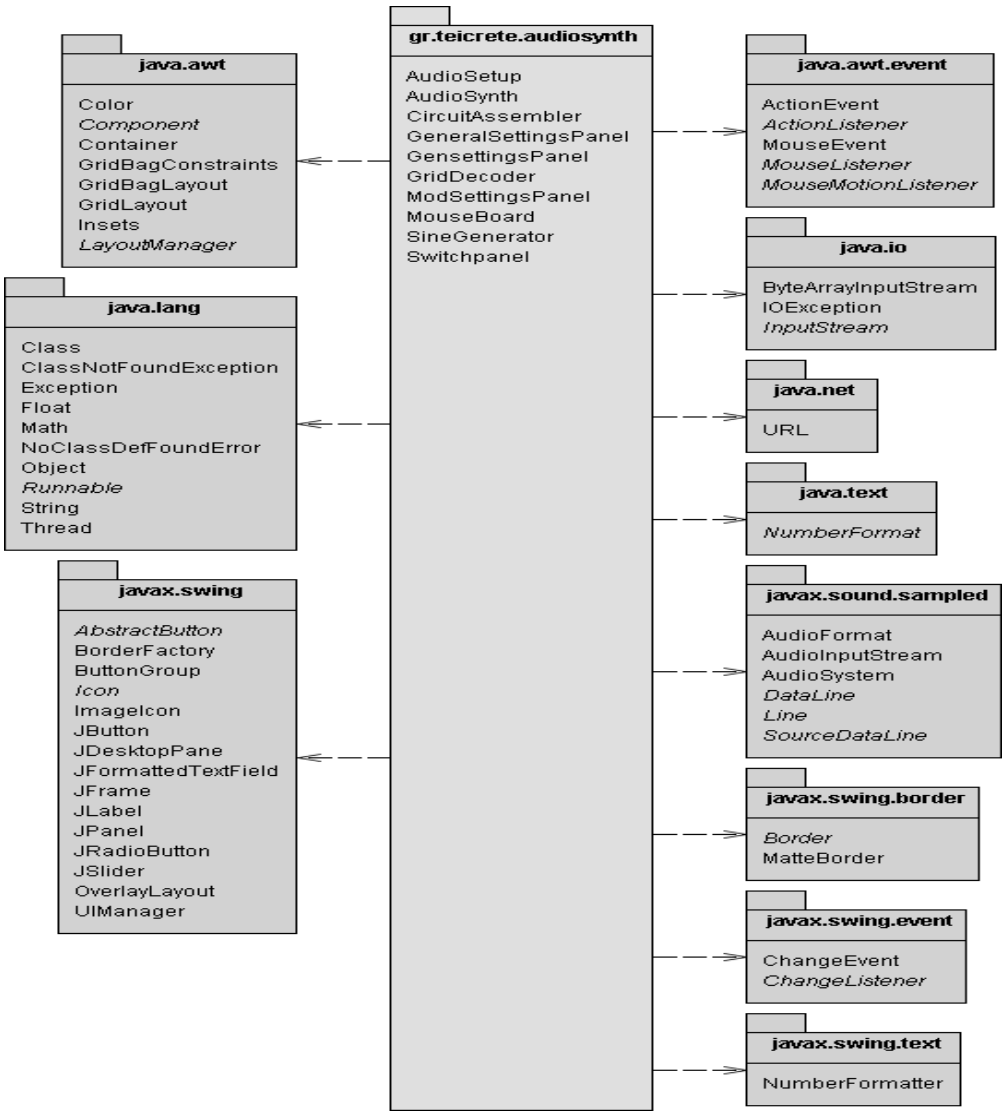
4. Η προσθήκη διαφόρων φίλτρων ή τεχνικών προσομοίωσης χώρου (reverb, delay κ.τ.λ.) τα οποία θα επέκτειναν τις ηχοχρωματικές δυνατότητες της εφαρμογής.

5. Τέλος, η δημιουργία νέων γραφικών συστατικών και η αντικατάσταση των default που παρέχονται από τις βιβλιοθήκες της Java, θα οδηγούσε σε μια αισθητικά αρτιότερη εμφάνιση της εφαρμογής.

# Παράρτημα: Τεκμηρίωση Κώδικα

Όπως έχει ήδη αναφερθεί η εφαρμογή AudioSynth δημιουργήθηκε με την Java 2 SDK 1.4 Edition. Η Java είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού και επομένως και η εφαρμογή AudioSynth μπορεί να εξεταστεί σαν ένα πλήθος αντικειμένων τα οποία συνδέονται και επικοινωνούν με διάφορους τρόπους μεταξύ τους. Τα αντικείμενα αυτά αποτελούν υποστάσεις διαφόρων κλάσεων. Οι κλάσεις αποτελούν μπλοκ κώδικα στα οποία ορίζονται τα γενικά χαρακτηριστικά και οι συμπεριφορές των διαφόρων αντικειμένων.

Η εφαρμογή AudioSynth, όπως φαίνεται και από το παρακάτω διάγραμμα, αποτελείται από 10 κλάσεις:




Εικόνα Π1

Στο μεσαίο πλαίσιο του διαγράμματος της εικόνας Π1 εμφανίζονται όλες οι κλάσεις του πακέτου gr.teicrete.audiosynth στο οποίο ανήκει εφαρμογή. Τα υπόλοιπα πλαίσια εμφανίζουν κλάσεις και interfaces που είναι ενσωματωμένα στην Java και χρησιμοποιούνται από την εφαρμογή.

Η κύρια κλάση της εφαρμογής είναι η **AudioSynth**. Σε αυτήν περιέχεται και η κύρια μέθοδος (main method). Ολόκληρη η εφαρμογή AudioSynth μπορεί να θεωρηθεί ως ένα αντικείμενο της κλάσης αυτής. Τα περισσότερα από τα αντικείμενα των άλλων κλάσεων δημιουργούνται σε αυτήν την κλάση και προστίθενται έτσι στην εφαρμογή του AudioSynth. Στην κλάση AudioSynth ορίζεται και ο τρόπος με τον οποίο τα κύρια αντικείμενα της εφαρμογής συνεργάζονται μεταξύ τους. Ο ορισμός αυτής της κλάσης γίνεται στο αρχείο με το όνομα AudioSynth.java.

Μέσα στην κλάση AudioSynth δημιουργούνται εννέα αντικείμενα της κλάσης **GensettingsPanel**. Αυτά τα αντικείμενα αποτελούν τα πλαίσια ρυθμίσεων ταλαντωτών. Όπως έχει αναφερθεί, κάθε στιγμή εμφανίζεται στην οθόνη του χρήστη ένα μόνο από τα πλαίσια αυτά. Η κλάση GensettingsPanel περιέχει προφανώς και μεθόδους με τις οποίες εξάγονται οι τιμές των διαφόρων ρυθμίσεων ώστε να χρησιμοποιηθούν στη σύνθεση ήχου.

Κάτω από το πλαίσιο ρυθμίσεων ταλαντωτών υπάρχει το πλαίσιο καθορισμού τύπων διαμόρφωσης. Αυτό το πλαίσιο αποτελεί ένα αντικείμενο της κλάσης **ModSettingsPanel** η οποία ορίζεται στο αρχείο ModSettingsPanel.java. Το πλαίσιο αυτό περιέχει δύο διακόπτες που ορίζουν τον τύπο των διαμορφώσεων. Έτσι αυτή η κλάση ορίζει και τον τρόπο ώστε αυτές οι ρυθμίσεις να στέλνονται στην κλάση AudioSynth και να λαμβάνονται υπ' όψιν κατά την διαδικασία της σύνθεσης ήχου. Οι ετικέτες 'AM' και 'FM' των αντίστοιχων διακοπτών είναι δυο αρχεία gif. Τα δύο αυτά αρχεία (AM.gif, FM.gif) πρέπει να είναι διαθέσιμα προκειμένου να εμφανίζεται σωστά το συγκεκριμένο πλαίσιο.

Το επόμενο πλαίσιο της εφαρμογής είναι αυτό που περιέχει το πλέγμα ταλαντωτών. Το πλέγμα ταλαντωτών αποτελεί αντικείμενο της κλάσης **Switchpanel** ο ορισμός της οποίας βρίσκεται στο αρχείο Switchpanel.java. Η κλάση αυτή πέρα από την εμφάνιση του grid, καθορίζει και τον τρόπο με τον οποίο εξάγονται οι διάφορες καταστάσεις του πλέγματος, όπως αυτές καθορίστηκαν από τον χρήστη. Η απεικόνιση των δύο καταστάσεων των διακοπτών που δημιουργούν το πλέγμα γίνεται με τη χρήση δύο gif αρχείων (grid.gif, gridselected.gif).

Το πλαίσιο των γενικών ρυθμίσεων που αποτελεί και το τελευταίο πλαίσιο του κύριου παράθυρου της εφαρμογής είναι αντικείμενο της κλάσης **GeneralSettingsPanel**. Το μοναδικό αντικείμενο αυτής της κλάσης δημιουργείται όπως και στις προηγούμενες περιπτώσεις στην κλάση AudioSynth. Αυτή η κλάση ορίζει την εμφάνιση του μοναδικού της αντικειμένου και τον τρόπο με τον οποίο οι ρυθμίσεις στο ανάλογο πλαίσιο θα είναι προσβάσιμες και από τα άλλα αντικείμενα της εφαρμογής. Ο ορισμός αυτής της κλάσης γίνεται στο αρχείο GeneralSettingsPanel.java.

Η τελευταία κλάση η οποία χρησιμοποιείται για την δημιουργία αντικειμένων τα οποία σχετίζονται με γραφικά είναι η κλάση **MouseBoard** η οποία βρίσκεται στο αρχείο MouseBoard.java. Είναι προφανές ότι το μοναδικό αντικείμενο αυτής της κλάσης που υλοποιείται στην εφαρμογή AudioSynth είναι το παράθυρο Mouseboard . Στην κλάση αυτή ορίζεται η εμφάνιση του αντίστοιχου παραθύρου, ο τρόπος με τον οποίο δημιουργούνται οι τιμές πλάτους και συχνότητας ανάλογα με την κίνηση του κέρσορα μέσα σε αυτό, καθώς και ο τρόπος γνωστοποίησης αυτών των τιμών προς τα υπόλοιπα αντικείμενα της εφαρμογής.

Μέχρι τώρα εξετάστηκαν κλάσεις και αντικείμενα τα οποία σχετίζονται με το interface του AudioSynth. Ωστόσο, η εφαρμογή AudioSynth περιλαμβάνει και υλοποιήσεις άλλων τεσσάρων κλάσεων οι οποίες αφορούν στον παραγόμενο ήχο. Φυσικά, τα αντικείμενα των κλάσεων που αφορούν στα γραφικά και τα αντικείμενα που σχετίζονται με τον ήχο, επικοινωνούν με διάφορους τρόπους μεταξύ τους.

Η πρώτη κλάση που σχετίζεται με την παραγωγή ήχου είναι η κλάση **SineGenerator**, ο ορισμός της οποίας γίνεται στο αρχείο SineGenerator.java. Όπως υποδηλώνει και το όνομα της, τα αντικείμενα αυτής της κλάσης είναι οι ημιτονικοί ταλαντωτές που χρησιμοποιούνται για την σύνθεση ήχου με την εφαρμογή AudioSynth. Έτσι δημιουργούνται εννέα αντικείμενα αυτής της κλάσης μέσα στον ορισμό της κλάσης AudioSynth.

Η κλάση **CircuitAssembler** που ορίζεται στο αρχείο CircuitAssembler.java, είναι η δεύτερη κλάση που σχετίζεται με την σύνθεση ήχου. Σε αυτήν την κλάση ορίζονται ‘πράξεις’ της σύνθεσης ήχου όπως η προσθετική σύνθεση σημάτων και η διαμόρφωση ταλαντωτών. Η κλάση αυτή προϋποθέτει, όπως είναι εμφανές, την ύπαρξη αντικειμένων της κλάσης SineGenerator στην οποία αναφερθήκαμε παραπάνω.

Οι διάφορες ‘πράξεις’ της σύνθεσης ήχου, γίνονται εφικτές με την ύπαρξη ενός αντικείμένου της κλάσης CircuitAssembler. Ωστόσο οι πράξεις αυτές υλοποιούνται μέσα σε μια από τις μεθόδους της κλάσης **GridDecoder** που βρίσκεται στο αρχείο GridDecoder.java. Η κλάση αυτή ελέγχει την κατάσταση του πλέγματος ταλαντωτών όπως αυτή δίνεται από μια μέθοδο της κλάσης SwitchPanel και αναλόγως χρησιμοποιεί τους εννέα ταλαντωτές της εφαρμογής για τη σύνθεση ήχου. Είναι εμφανές ότι στον ορισμό αυτής της κλάσης θα υπάρχουν πολλές αναφορές σε αντικείμενα των παραπάνω κλάσεων.

Το μοναδικό αντικείμενο της κλάσης GridDecoder, φέρει σε πέρας την διαδικασία της σύνθεσης ήχου. Αυτό που απομένει είναι ο ήχος που έχει παραχθεί σε αυτό το σημείο, που δεν είναι παρά κάποιες αριθμητικές τιμές που επιστρέφονται από μια μέθοδο του συγκεκριμένου αντικείμενου, να μεταβιβαστεί στην κάρτα ήχου του υπολογιστή ώστε να αναπαραχθεί. Αυτός ακριβώς είναι ο ρόλος του μοναδικού αντικειμένου της κλάσης **AudioSetup**. Σε αυτήν την κλάση ορίζεται ποιο θα είναι το format του προς αναπαραγωγή ήχου, επιτυγχάνεται η επικοινωνία με την κάρτα ήχου και τελικά οι αριθμητικές τιμές για τις οποίες μιλήσαμε παραπάνω, μετατρέπονται σε μια διαρκώς μεταβαλλόμενη ακολουθία byte που τελικά στέλνεται στην κάρτα ήχου, ώστε να

αναπαραχθεί ο ήχος. Ο ορισμός της κλάσης AudioSetup γίνεται στο αρχείο AudioSetup.java.

Ακολουθεί μία πιο αναλυτική περιγραφή των παραπάνω κλάσεων.

## i. Η κλάση AudioSynth

Παρακάτω βρίσκεται το πρώτο μπλοκ κώδικα της κλάσης AudioSynth που βρίσκεται, όπως ήδη έχει αναφερθεί, στο αρχείο AudioSynth.java.

```
package gr.teicrete.audiosynth;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JDesktopPane;
import javax.swing.OverlayLayout;
import javax.swing.UIManager;

import java.awt.Color;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
```

Στην πρώτη γραμμή βρίσκεται τη δήλωση του πακέτου στο οποίο συμπεριλαμβάνεται η κλάση. Οι κλάσεις στην Java μπορεί να οργανώνονται σε πακέτα, το όνομα των οποίων ορίζεται από τον χρήστη. Η δήλωση πακέτων με ονόματα που αντιστοιχούν σε αντίστροφη γραφή θέσεων ιστοσελίδων αποτελεί σύμβαση που είναι αποδεκτή από πολλούς Java προγραμματιστές. Όλες οι κλάσεις της εφαρμογής AudioSynth βρίσκονται στο πακέτο με όνομα gr.teicrete.audiosynth.

Στις παρακάτω γραμμές του κώδικα δηλώνονται ποιες από τις κλάσεις που είναι ενσωματωμένες στην συγκεκριμένη έκδοση της Java χρησιμοποιούνται στον ορισμό της κλάσης AudioSynth. Αυτό γίνεται με τη δήλωση import πριν από την πλήρη διαδρομή και το όνομα της κλάσης. Για παράδειγμα η δήλωση import javax.swing.JFrame; κάνει εφικτή τη χρήση της κλάσης JFrame που βρίσκεται στο πακέτο javax.swing.

Η κλάση JFrame ορίζει τα βασικά χαρακτηριστικά για τον σχεδιασμό ενός παραθύρου με τη Java. Η κλάσεις JPanel και JDesktopPane αφορούν σε panel τα οποία συναντώνται με διάφορους τρόπους διάταξης μέσα σε ένα παράθυρο. Οι κλάσεις OverlayLayout, GridBagLayout και GridBagConstraints αφορούν στον τρόπο που διατάσσονται τα συστατικά μέσα σε ένα panel. Η κλάση UIManager αφορά στο στυλ εμφάνισης μιας εφαρμογής ενώ η κλάση Color χρησιμεύει στην χρήση έτοιμων χρωμάτων καθώς και για τη δημιουργία νέων.

Στην επόμενη σειρά του κώδικα δίνεται το όνομα της κλάσης και ορισμένα χαρακτηριστικά αυτής.

```
public class AudioSynth extends JFrame implements  Runnable {
```

Η δήλωση public αφορά στην πρόσβαση σε αυτήν την κλάση. Δηλώνει συγκεκριμένα ότι ο προγραμματιστής μπορεί να έχει πρόσβαση σε αυτήν την κλάση από οποιαδήποτε κλάση του ίδιου πακέτου χωρίς να χρειαστεί να την εισάγει με τη δήλωση import. Η φράση extends JFrame σημαίνει ότι η συγκεκριμένη κλάση αποτελεί ένα παράθυρο και επομένως επεκτείνει την κλάση JFrame που όπως αναφέρθηκε περιγράφει τα γενικά χαρακτηριστικά ενός παραθύρου στην Java. Η φράση implements Runnable σημαίνει ότι η συγκεκριμένη κλάση πραγματοποιεί τη διασύνδεση Runnable η οποία αφορά στη δημιουργία νημάτων (Threads) τα οποία θα εξετάστουν παρακάτω. Μια κλάση μπορεί να αποτελεί επέκταση μόνο μιας άλλης η οποία ονομάζεται υπερκλάση, αλλά μπορεί να πραγματοποιεί πολλαπλές διασυνδέσεις επιτυγχάνοντας έτσι την ύπαρξη «πολλαπλής κληρονομικότητας».

Το άγκιστρο που βρίσκεται στο τέλος της παραπάνω γραμμής δηλώνει ότι σε αυτό το σημείο ξεκινάει ο ορισμός της κλάσης. Όλες οι κλάσεις, οι μέθοδοι και οι διάφοροι βρόχοι κώδικα βρίσκονται μέσα σε τέτοια άγκιστρα.

Το επόμενο κομμάτι κώδικα φαίνεται παρακάτω και ονομάζεται μπλοκ αρχικοποίησης (Initialization Block).

```
    protected static int VISIBLEGSPANEL=0;
protected static int MODULATION1TYPE=0;//αρχική πρώτη διαμόρφωση: AM
protected static int MODULATION2TYPE=0;//αρχική δεύτερη διαμόρφωση: AM Modulation
protected static int GRIDSTATE = 0;    //αρχική κατάσταση του Grid
protected static boolean isMovingonPlaypanel = false; //boolean μεταβλητή που δηλώνει ότι ο κέρσορας
                                    //δεν βρίσκεται εντός του mouseboard
protected static boolean playing = false; //Audio off

protected static int Osc1ON = 0;
protected static int Osc2ON = 0;
protected static int Osc3ON = 0;
protected static int Osc4ON = 0; // Όλοι οι διακόπτες του Grid αρχικά
protected static int Osc5ON = 0; // είναι σε κατάσταση OFF
protected static int Osc6ON = 0;
protected static int Osc7ON = 0;
protected static int Osc8ON = 0;
protected static int Osc9ON = 0;

private double frequency; // πλάτος και συχνότητα του GensettingsPanel
private double amplitude; // που εμφανίζεται στην οθόνη του χρήστη

protected double [] osc1settings = new double [3] ;
protected double [] osc2settings = new double [3] ;
protected double [] osc3settings = new double [3] ;
protected double [] osc4settings = new double [3] ; // Δημιουργία πινάκων 3 θέσεων
protected double [] osc5settings = new double [3] ; // που θα αποθηκεύουν τις τιμές
protected double [] osc6settings = new double [3] ; // του πλάτους της συχνότητας και
protected double [] osc7settings = new double [3] ; // του δείκτη διαμόρφωσης για
protected double [] osc8settings = new double [3] ; // κάθε ταλαντωτή
protected double [] osc9settings = new double [3] ;
protected double [] visibleoscsettings = new double [3] ;

SineGenerator sine1osc = new SineGenerator();
```

```
SineGenerator sine2osc = new SineGenerator();
SineGenerator sine3osc = new SineGenerator();
SineGenerator sine4osc = new SineGenerator();
SineGenerator sine5osc = new SineGenerator();   //Δημιουργία των 9 ταλαντωτών
SineGenerator sine6osc = new SineGenerator();
SineGenerator sine7osc = new SineGenerator();
SineGenerator sine8osc = new SineGenerator();
SineGenerator sine9osc = new SineGenerator();

GridDecoder  GD1 = new GridDecoder (); //Δημιουργία του αποκωδικοποιητή του Grid

private static AudioSynth me = null;  // Αρχικοποίηση εφαρμογής

MouseBoard mouseboard = new MouseBoard(); //Δημιουργία του Mouseboard

JDesktopPane contentpanel = new JDesktopPane();
JPanel gridreceiverpanel = new JPanel();        //Δημιουργία κάποιων panel
JPanel sinepanelsreceiver = new JPanel();       //που θα χρησιμοποιηθούν στην δημιουργία
JPanel componentspanel = new JPanel();          //του Layout της εφαρμογής

AudioSetup g = new AudioSetup();
GensettingsPanel sinesettingspanel_00     = new GensettingsPanel("SINE (1,1)");// Δημιουργία των 9
                                                                                //πλαισίων
GensettingsPanel sinesettingspanel_01     = new GensettingsPanel("SINE (1,2)");// ρυθμίσεων ταλαντωτών
GensettingsPanel sinesettingspanel_02     = new GensettingsPanel("SINE (1,3)");//
GensettingsPanel sinesettingspanel_10     = new GensettingsPanel("SINE (2,1)");
GensettingsPanel sinesettingspanel_11     = new GensettingsPanel("SINE (2,2)");
GensettingsPanel sinesettingspanel_12     = new GensettingsPanel("SINE (2,3)");
GensettingsPanel sinesettingspanel_20     = new GensettingsPanel("SINE (3,1)");
GensettingsPanel sinesettingspanel_21     = new GensettingsPanel("SINE (3,2)");
GensettingsPanel sinesettingspanel_22     = new GensettingsPanel("SINE (3,3)");
GensettingsPanel vsinesettingspanel=sinesettingspanel_00;  // μεταβλητή τύπου GensettingsPanel
                                    //που θα αντιστοιχεί στο GensettingsPanel που είναι ορατό

ModSettingsPanel modsettingspanel = new ModSettingsPanel();  //Δημιουργία του ModSettingsPanel
     GeneralSettingsPanel generalsettingspanel = new GeneralSettingsPanel(); //Δημιουργία του
                                                                             //GeneralSettingsPanel
Switchpanel switchpanel = new Switchpanel();  //Δημιουργία του SwitchPanel

private static final int theseis = 1048576;                  //μεταβλητές που χρησιμοποιούνται
protected static final float [] sinetable = new  float[theseis];//για την δημιουργία πίνακα 1048576 θέσεων
private double step;                                  //από όπου οι ταλαντωτές διαβάζουν τις τιμές
private double x;                                    //ενός ημιτονικού κύματος

static protected Color mygreen = new Color(120,220,105);// δημιουργία του πράσινου ανοιχτου χρώματος
                                  //που χρησιμοποιείται στην εφαρμογή.
```

Στο παραπάνω μπλοκ κώδικα αρχικοποιούνται κάποιες μεταβλητές και αντικείμενα που θα χρησιμοποιηθούν αργότερα στον ορισμό της κλάσης. Δεν θα εξετάσουμε σε αυτό το σημείο αναλυτικά τον κώδικα αφού αργότερα θα εξηγηθεί αναλυτικά η χρησιμότητα αυτών των μεταβλητών και αντικειμένων. Ωστόσο, ο παραπάνω κώδικας συνοδεύεται από σχόλια στην ελληνική γλώσσα που μπορεί να είναι κατατοπιστικά για τον αναγνώστη. Τα σχόλια στον Java προγραμματισμό εισάγονται με τα σύμβολα //. Όταν υπάρχουν αυτά τα σύμβολα σε μια γραμμή κώδικα, οτιδήποτε ακολουθεί αποτελεί

σχόλιο και δεν το λαμβάνει υπ' όψιν ο επεξεργαστής του υπολογιστή όταν κάνει compile την εφαρμογή. Ένας άλλος τρόπος εισαγωγής σχολίων προϋποθέτει την ύπαρξη των συμβόλων /* και */. Όσες γραμμές κώδικα βρίσκονται ανάμεσα σε αυτά τα δύο σύμβολα αναγνωρίζονται από τον επεξεργαστή σαν σχόλια.

Η επόμενη γραμμή κώδικα είναι η αρχή της μεθόδου δημιουργού της κλάσης AudioSynth και φαίνεται παρακάτω:

```
private AudioSynth () {
```

Αυτήν την μέθοδο πρέπει να καλέσει ο προγραμματιστής για να δημιουργήσει ένα αντικείμενο της κλάσης AudioSynth. Η μέθοδος αυτή ορίζεται ως private. Αυτό σημαίνει ότι μπορεί να κληθεί μόνο μέσα από τη συγκεκριμένη κλάση.

Ακολουθούν οι επόμενες γραμμές κώδικα της μεθόδου.

```
                super("AudioSynth");
setLocation (5,5);
setSize(380,375);
setResizable(false);
setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
setContentPane (contentpanel);
GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
```

Η πρώτη γραμμή καλεί τη μέθοδο δημιουργό της υπερκλάσης της κλάσης AudioSynth δηλαδή της κλάσης JFrame. Το String "AudioSynth" που βρίσκεται εντός των παρενθέσεων προορίζεται για τίτλος του παραθύρου. Οι τρεις επόμενες γραμμές καθορίζουν το μέγεθος και την θέση του παραθύρου καθώς και το ότι ο χρήστης δεν μπορεί να επέμβει στο μέγεθος του παραθύρου. Στη συνέχεια ορίζεται ποιο από τα panel είναι αυτό που θα περιλαμβάνει όλα τα υπόλοιπα.

Οι δύο τελευταίες γραμμές αφορούν στον τρόπο διάταξης των συστατικών σε ένα πλαίσιο ή παράθυρο. Παρόλο που δεν θααναφερθούν λεπτομέρειες σχετικά με τα γραφικά σημειώνεται ότι οι διαχειριστές διάταξης είναι κλάσεις οι οποίες ορίζουν κάποιους γενικούς κανόνες για την διάταξη των συστατικών. Η κλάση GridBagLayout δημιουργεί πλέγματα συστατικών το κάθε κελί των οποίων μπορεί να έχει διαφορετικό μέγεθος από το άλλο.

```
for(int i=0; i<theseis; i++) {
        step = ((2.0 * Math.PI) /theseis);
        sinetable[i] =  (float) (Math.sin(x));
        x=x+step;
}
```

Ο βρόχος for που φαίνεται παραπάνω και που αποτελεί τη συνέχεια του κώδικα, γεμίζει με τιμές τις 1048576 θέσεις του πίνακα sinetable[]. Ορίζεται σαν βήμα εγγραφής στον πίνακα ίσο με 2*π / θέσεις πίνακα. Αυτό σε κάθε κύκλο του βρόχου προστίθεται στη μεταβλητή x που όπως μπορούμε να καταλάβουμε από την τελευταία γραμμή του βρόχου θα παίρνει τελικά τιμές από 0 ως 2*π. Έτσι καταφέρνουμε ο πίνακας sinetable[]

να παίρνει τις τιμές τις συνάρτησης του ημιτόνου για το διάστημα από 0 ως 2π. Σε αυτόν τον πίνακα θα γίνει αναφορά μέσα από την κλάση SineGenerator αφού πρόκειται για τον πίνακα από όπου διαβάζουν τις τιμές τους όλοι οι ταλαντωτές της εφαρμογής.

```
OverlayLayout over1 = new OverlayLayout(contentpanel);
OverlayLayout over2 = new OverlayLayout(sinepanelsreceiver);
contentpanel.setLayout(over1);
componentspanel.setLayout(gridbag);
sinepanelsreceiver.setLayout(over2);
```

Αυτές οι γραμμές κώδικα αφορούν και πάλι στους διαχειριστές διάταξης. Σημειώνεται μόνο ότι ο διαχειριστής gridbag της κλάσης GridBagLayout για την οποία μιλήσαμε παραπάνω αφορά στο JPanel αντικείμενο componentspanel το οποίο είναι αυτό που θα περιέχει όλα τα αντικείμενα των άλλων κλάσεων γραφικών (GeneralSettingsPanel, ModSettingsPanel κ.τλ.). Το JPanel αντικείμενο sinepanelsreceiver στο οποίο αργότερα θα προστεθούν όλα τα πλάισια ρυθμίσεων ταλαντωτών έχει διαχειριστή διάταξης της κλάσης OverLayLayout. Αυτό σημαίνει ότι όλα τα αντικείμενα που θα τοποθετηθούν σε αυτό το πλαίσιο θα αλληλοεπικαλύπτονται.

```
sinepanelsreceiver.add(sinesettingspanel_00);
sinepanelsreceiver.add(sinesettingspanel_01);
sinepanelsreceiver.add(sinesettingspanel_02);
sinepanelsreceiver.add(sinesettingspanel_10);
sinepanelsreceiver.add(sinesettingspanel_11);
sinepanelsreceiver.add(sinesettingspanel_12);
sinepanelsreceiver.add(sinesettingspanel_20);
sinepanelsreceiver.add(sinesettingspanel_21);
sinepanelsreceiver.add(sinesettingspanel_22);

sinesettingspanel_00.setVisible(true);
sinesettingspanel_01.setVisible(false);
sinesettingspanel_02.setVisible(false);
sinesettingspanel_10.setVisible(false);
sinesettingspanel_11.setVisible(false);
sinesettingspanel_12.setVisible(false);
sinesettingspanel_20.setVisible(false);
sinesettingspanel_21.setVisible(false);
sinesettingspanel_22.setVisible(false);

gridreceiverpanel.add(switchpanel);
```

Στις πρώτες εννιά γραμμές του παραπάνω κώδικα γίνεται η πρόσθεση των πλαισίων ρυθμίσεων ταλαντωτών στο sinepanelsreceiver. Στις επόμενες εννιά γραμμές του κώδικα γίνεται ορατό στον χρήστη είναι μόνο το πλαίσιο ρυθμίσεων που αφορά στον ταλαντωτή 1. Υπενθυμίζεται ότι η μέθοδος δημιουργός της κλάσης AudioSynth καλείται όταν εκτελείται η εφαρμογή και επομένως ο κώδικας αφορά μόνο στην αρχική μορφή που θα έχει το πρόγραμμα. Τέλος στην επόμενη γραμμή όπως ήταν αναμενόμενο στο αντικείμενο gridreceiverpanel θα προστεθεί το switchpanel το οποίο είναι αντικείμενο της κλάσης SwitchPanel.

```
//Add SinePanels
```

```
buildConstraints(constraints, 0, 0, 1, 5, 100,5);
constraints.fill = GridBagConstraints.HORIZONTAL;
 constraints.anchor = GridBagConstraints.NORTH;
 gridbag.setConstraints(sinepanelsreceiver, constraints);
 componentspanel.add(sinepanelsreceiver);
//Add modsettingsPanel
buildConstraints(constraints, 0, 5, 1, 1, 100,5);
 constraints.fill = GridBagConstraints.HORIZONTAL;
        constraints.anchor = GridBagConstraints.NORTH;
 gridbag.setConstraints(modsettingspanel, constraints);
 componentspanel.add(modsettingspanel);
//Add gridreceiverpanel
buildConstraints(constraints, 0, 6, 1, 2, 100,5);
       constraints.fill = GridBagConstraints.HORIZONTAL;
       constraints.anchor = GridBagConstraints.NORTH;
       gridbag.setConstraints(gridreceiverpanel, constraints);
gridreceiverpanel.setBackground(mygreen);
gridreceiverpanel.setForeground(mygreen);
 componentspanel.add(gridreceiverpanel);
//Add generalsettingspanel
buildConstraints(constraints, 0, 8, 1, 2, 100,5);
 constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.anchor = GridBagConstraints.PAGE_START;
 gridbag.setConstraints(generalsettingspanel, constraints);
 componentspanel.add(generalsettingspanel);
contentpanel.add(componentspanel);

}
```

Οι παραπάνω γραμμές κώδικα αποτελούν τις τελευταίες της μεθόδου δημιουργού της κλάσης AudioSynth όπως υποδηλώνεται και από το άγκιστρο που βρίσκεται κάτω από αυτές. Ο κώδικας αυτός αφορά στον τρόπο που θα διαταχθούν τα συστατικά πάνω στο πλαίσιο componentspanel . Η εξέταση αυτών των λεπτομερειών σχετικά με τα γραφικά δεν είναι στους σκοπούς αυτής της εργασίας. Επισημαίνεται ωστόσο ότι πριν την πρόσθεση των συστατικών γίνεται κλίση στην μέθοδο setConstraints(JComponent comp, GridBagConstraints constr). Η μέθοδος αυτή βρίσκεται μέσα στην κλάση AudioSynth και φαίνεται παρακάτω . Ο κώδικας που ακολουθεί είναι ακριβής αντιγραφή από το πλήρες εγχειρίδιο της Java2 των Rogers Cadenhead και Laura Lemay (ελληνική έκδοση Μ.Γκιούρδας).

```
protected static final void buildConstraints(GridBagConstraints gbc, int gx, int gy,
int gw, int gh, int wx, int wy){
gbc.gridx = gx;
gbc.gridy = gy;
gbc.gridwidth = gw;
gbc.gridheight = gh;
gbc.weightx = wx;
gbc.weighty = wy;
}
```

Η παραπάνω μέθοδος έχει δηλωθεί protected. Αυτό σημαίνει ότι μπορεί να κληθεί από όλες τις κλάσεις μέσα στο ίδιο πακέτο. Κατά την εξέταση και των άλλων κλάσεων

θα δούμε ότι γίνεται κλήση σε αυτή τη μέθοδο αρκετές φορές προκειμένου να καθορίσουμε την διάταξη των συστατικών μας.

```
public static AudioSynth getInstance() {
if( me == null) {
   me = new AudioSynth();
}
return me;
}
```

Η παραπάνω μέθοδος επιστρέφει ένα αντικείμενο AudioSynth. Αν το αντικείμενο me που φαίνεται στο μπλοκ αρχικοποίησης έχει την τιμή "null" τότε δημιουργείται ένα νέο αντικείμενο AudioSynth με αυτό το όνομα. Αν το αντικείμενο me υπάρχει ήδη, τότε η μέθοδος getInstance() επιστρέφει αυτό ακριβώς το αντικείμενο. Η μέθοδος έχει δηλωθεί static. Αυτό σημαίνει ότι μπορούμε να καλέσουμε αυτήν την μέθοδο χωρίς προηγουμένως να έχει δημιουργηθεί κανένα αντικείμενο της κλάσης AudioSynth. Θα δούμε κατά την εξέταση των άλλων κλάσεων ότι όταν θέλουμε να αναφερθούμε σε μία μέθοδο ή μια μεταβλητή υπόστασης της κλάσης AudioSynth, θα καλούμε αυτήν την μέθοδο. Στην περίπτωση που θα θέλουμε να αναφερθούμε σε μία static μεταβλητή ή μέθοδο δεν υπάρχει λόγος να καλέσουμε αυτήν την μέθοδο αφού αυτές δεν αφορούν σε ένα αντικείμενο μιας κλάσης.

Η επόμενες δύο μέθοδοι της κλάσης AudioSynth φαίνονται παρακάτω.

```
public void play() {
Thread thread1 = new Thread(this);
thread1.start();
}

public void run () {
while(playing) {
      g.writeBytestoLine(g.buildAudioBytes(GD1));
}
}
```

Στην πρώτη μέθοδο play() δημιουργείται το αντικείμενο thread1 της κλάσης Thread. Η χρήση νημάτων εκτέλεσης (Threads) είναι εφικτή για την κλάση AudioSynth αφού όπως είδαμε αυτή υλοποιεί την διασύνδεση Runnable. Τα νήματα εκτέλεσης χρησιμοποιούνται για να κατανέμουν έξυπνα το χρόνο εκτέλεσης του επεξεργαστή, ώστε να φαίνεται ότι αυτός κάνει περισσότερα από ένα πράγματα ταυτόχρονα. Στην συγκεκριμένη περίπτωση χρησιμοποιείται αυτό το thread ώστε ο επεξεργαστής να αναπαράγει ήχο και ταυτόχρονα να μπορούμε να αλλάζουμε τις ρυθμίσεις στο interface του οργάνου. Στην πραγματικότητα αυτό δεν γίνεται ταυτόχρονα. Όποιες άλλες εργασίες που εκτελεί ο επεξεργαστής θα γίνουν στο χρονικό διάστημα που μεσολαβεί ανάμεσα στην αναπαραγωγή δύο ηχητικών δειγμάτων. Στην συγκεκριμένη περίπτωση η συχνότητα δειγματοληψίας είναι 44100 samples/sec άρα το αυτό το χρονικό διάστημα θα ισούται με 1/44100 sec. Μόλις δημιουργηθεί το thread καλούμε για αυτό τη μέθοδο start. Αυτή μας η κίνηση μας καλεί αυτομάτως τη μέθοδο run().

Σε αυτήν τη μέθοδο δηλώνεται ποιες ενέργειες είναι επιθυμητό να εκτελούνται μέσα σε ξεχωριστό thread. Στη συγκεκριμένη περίπτωση ζητείται να επαναλαμβάνονται κάποιες ενέργειες για όσο η Boolean μεταβλητή "playing" είναι αληθής. Θα γίνει εμφανές αργότερα ότι αυτή η μεταβλητή είναι αληθής όποτε ο διακόπτης Audio του πλαισίου γενικών ρυθμίσεων είναι στη θέση ON. Οι μέθοδοι που καλούνται μέσα στον βρόχο while αφορούν στο αντικείμενο g της κλάσης AudioSetup. Αναφέρεται απλά ότι η μία από αυτές τις μεθόδους δημιουργεί μια ακολουθία από byte, ενώ η άλλη «γράφει» αυτά τα bytes στην κάρτα ήχου.

```
protected void updateOscillatorsSettings(){
  switch(AudioSynth.GRIDSTATE){
        case 100:
              sinesettingspanel_00.setModulatorPanel(2);
              sinesettingspanel_01.setModulatorPanel(2);
              sinesettingspanel_02.setModulatorPanel(2);
              break;
        case 10:
              sinesettingspanel_10.setModulatorPanel(2);
              sinesettingspanel_11.setModulatorPanel(2);
              sinesettingspanel_12.setModulatorPanel(2);
              break;
……..
……..
        case 1:
              AudioSynth.getInstance().sinesettingspanel_20.setModulatorPanel(2);
              AudioSynth.getInstance().sinesettingspanel_21.setModulatorPanel(2);
              AudioSynth.getInstance().sinesettingspanel_22.setModulatorPanel(2);
              break;

  }
  updateallindexes();
}
```

Η παραπάνω μέθοδος με το όνομα updateOscillatorSettings καλείται όποτε αλλάζει η κατάσταση του πλέγματος ταλαντωτών ή οι τύποι διαμόρφωσης στο σχετικό πλαίσιο. Σκοπός αυτής της μεθόδου είναι να αλλάζει την μορφή των πλαισίων ρυθμίσεων ταλαντωτών ανάλογα με το αν αυτοί είναι διαμορφωτές και τι είδους. Η πρώτη περίπτωση όπου η ακέραια μεταβλητή GRIDSTATE έχει την τιμή 100 αφορά στην περίπτωση όπου σε κατάσταση ON είναι ένας τουλάχιστον διακόπτης της πρώτης γραμμής του grid και κανένας στις άλλες δύο γραμμές. Όταν εξετάστούν οι άλλες κλάσεις της εφαρμογής θα γίνει κατανοητός ο τρόπος με τον οποίο αλλάζει τιμές η μεταβλητή GRIDSTATE, ενώ για οικονομία χώρου παραπάνω δεν φαίνεται ολόκληρη η μέθοδος. Στο τέλος της παραπάνω μεθόδου έχουμε κλήση στη μέθοδο updateallindexes(). Η μέθοδος αυτή φαίνεται παρακάτω:

```
protected void updateallindexes(){
osc1settings[2]=this.sinesettingspanel_00.indexMultiplied;
osc2settings[2]=this.sinesettingspanel_01.indexMultiplied;
osc3settings[2]=this.sinesettingspanel_02.indexMultiplied;
osc4settings[2]=this.sinesettingspanel_10.indexMultiplied;
```

```
        osc5settings[2]=this.sinesettingspanel_11.indexMultiplied;
        osc6settings[2]=this.sinesettingspanel_12.indexMultiplied;
    }
```

Με αυτή την μέθοδο αποθηκεύονται στους ανάλογους πίνακες οι τιμές οι σχετικές με τους δείκτες διαμόρφωσης. Είναι προφανές ότι και αυτή η μέθοδος καλείται στις ίδιες περιπτώσεις με την προηγούμενη. Αυτή η μέθοδος μας εξασφαλίζει ότι για παράδειγμα όταν ένας διαμορφωτής AM γίνεται FM ο δείκτης του θα παίρνει αντί για τιμές από 0 ως 1 , 0 ως 20.

```
        protected void setAmplitudeFrequencyandIndex(double Amp,double Freq,double Ind){
        switch(VISIBLEGSPANEL) {
            case 0:
                osc1settings[0]=Amp;
                osc1settings[1]=Freq;
                osc1settings[2]=Ind;
                break;
            case 1:
                osc2settings[0]=Amp;
                osc2settings[1]=Freq;
                osc2settings[2]=Ind;
                break;
……..
……..
        }

        }
```

Η παραπάνω μέθοδος καλείται όποτε αλλάζει κάποια από τις ρυθμίσεις του πλαισίου ρυθμίσεων ταλαντωτών. Ανάλογα με το ποιου ταλαντωτή οι ρυθμίσεις είναι ορατές (μεταβλητή VISIBLEGSPANEL) αποθηκεύονται οι τιμές της συχνότητας, του πλάτους και του δείκτη διαμόρφωσης στους ανάλογους πίνακες.

```
        protected void showGensettingsPanel() {
            switch(VISIBLEGSPANEL) {
             case 0:vsinesettingspanel.setVisible(false);
                sinesettingspanel_00.setVisible(true);
                vsinesettingspanel=sinesettingspanel_00;
                break;
……..
……..
            }
        }
```

Η παραπάνω μέθοδος showGensettingsPanel() καλείται όποτε κάνουμε δεξί κλικ σε κάποιον από τους διακόπτες του πλέγματος ταλαντωτών, ώστε να εμφανίζεται το ανάλογο πλαίσιο ρυθμίσεων ταλαντωτή. Όπως θα δούμε στην εξέταση της κλάσης SwitchPanel όποτε κάνουμε δεξί κλικ σε κάποιον διακόπτη αλλάζει η τιμή της integer μεταβλητής VISIBLEGSPANEL. Τότε μέσω της παραπάνω μεθόδου εξαφανίζεται το πλαίσιο ρυθμίσεων που ήταν πριν ορατό(vsinesettingspanel.setVisible(false);) και εμφανίζουμε το επιθυμητό πλαίσιο(sinesettingspanel_xx.setVisible(true);). Στην συνέχεια κατοχυρώνουμε στη

μεταβλητή vsinesettingspane το ορατό αντικείμενο, ώστε να μπορούμε να το εξαφανίσουμε με τον ίδιο τρόπο όποτε κάνουμε δεξί κλικ σε κάποιον άλλο διακόπτη.

Η παρακάτω μέθοδος με τον τίτλο updateSliderPosition καλείται όποτε έχουμε κίνηση μέσα στο Mouseboard, ώστε να ανανεώνονται οι ρυθμίσεις του πλάτους και της συχνότητας στο αντίστοιχο πλαίσιο ρυθμίσεων ταλαντωτή.

```
protected void updateSliderPosition(float xRatio, float yRatio) {
        frequency = (vsinesettingspanel.getMaximumFrequency()* ( Math.pow( 2.0, (xRatio)) - 1 )  );
    amplitude = 1 - ( yRatio);
    vsinesettingspanel.amplitude=1 - ( yRatio);
    vsinesettingspanel.frequency=this.frequency;
        vsinesettingspanel.ampGenSlider.setValue((int)
(amplitude*vsinesettingspanel.ampGenSlider.getMaximum()));
        vsinesettingspanel.frGenSlider.setValue((int)(xRatio*vsinesettingspanel.frGenSlider.getMaximum()));
    vsinesettingspanel.genAmpTextField.setValue(new Float(10*(Math.log(amplitude))));
    vsinesettingspanel.genFrTextField.setValue(new Float((float) (frequency)));
    switch(VISIBLEGSPANEL) {
        case 0:
            osc1settings[0]=amplitude;
            osc1settings[1]=frequency;
            break;
……..
……..
    }
    }
```

Η μέθοδος αυτή δέχεται σαν ορίσματα δύο float μεταβλητές xRatio και yRatio. Αυτές αφορούν στις συντεταγμένες του κέρσορα μέσα στο Mouseboard. Με βάση αυτές τις συντεταγμένες στις παρακάτω γραμμές ορίζονται οι νέες τιμές του πλάτους και της συχνότητας και ενημερώνονται τα sliders και τα πλαίσια εμφάνισης των τιμών αυτών. Στην πρώτη γραμμή της μεθόδου βλέπουμε ότι η συχνότητα είναι εκθετικά ανάλογη στο όρισμα xRatio που αφορά στη θέση του κέρσορα στον οριζόντιο άξονα του Mouseboard (Math.pow( 2.0, (xRatio)) - 1 )  = $2^{xRatio-1}$).

Στις παρακάτω γραμμές κώδικα βλέπουμε την τελευταία μέθοδο της κλάσης AudioSynth που αποτελεί και τη λεγόμενη «κύρια (main)» μέθοδο της εφαρμογής.

```
    public static void main (String [] args){
    try {
        UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        //UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        //UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
    } catch(Exception e) {
      e.printStackTrace();
    }
    AudioSynth vcs = AudioSynth.getInstance();
    vcs.show();


    }
}
```

Παρατηρούμε αρχικά έναν βρόχο try. Μέσα σε αυτόν τον βρόχο ορίζεται το στυλ που θα έχουν τα γραφικά. Το στυλ ορίζεται από αντικείμενα της κλάσης LookAndFeel. Το στυλ που δίνουμε στην εφαρμογή είναι το στάνταρτ στυλ για Java γραφικά και με αυτό το στυλ η εφαρμογή θα έχει την ίδια μορφή σε όλα τα λειτουργικά συστήματα στα οποία μπορεί να εκτελεστεί. Αν αντί για αυτή την εντολή δίναμε την εντολή που εμφανίζεται σαν σχόλιο ακριβώς από κάτω, η εφαρμογή θα έπαιρνε το στυλ του κάθε λειτουργικού συστήματος. Το δεύτερο σχόλιο αφορά σε ένα τρίτο στυλ εμφάνισης γραφικών με το όνομα Motif.

Το μπλοκ try που είδαμε καθώς και το μπλοκ catch που ακολουθεί αφορούν στις λεγόμενες «εξαιρέσεις» της Java. Οι εξαιρέσεις είναι ένα σύστημα διαχείρισης σφαλμάτων. Στην συγκεκριμένη περίπτωση θα είχαμε σφάλμα αν ορίζαμε η εφαρμογή να έχει την εμφάνιση που ορίζεται σε ένα LookAndFeel που δεν είναι εγκατεστημένο στο εκάστοτε σύστημα. Μέσα στο μπλοκ catch ορίζουμε τι πρέπει να γίνεται σε τέτοιες περιπτώσεις. Στην συγκεκριμένη περίπτωση εμφανίζουμε ένα στάνταρτ μήνυμα της Java σε DOS περιβάλλον. Αν πιστεύαμε ότι υπάρχει περίπτωση να συμβεί τέτοιο σφάλμα, ενδεχομένως να ορίζαμε να ανοίγει ένα παράθυρο το οποίο θα προειδοποιούσε τον χρήστη σχετικά.

Στις επόμενες σειρές δημιουργούμε την εφαρμογή με την κλήση στη μέθοδο getInstance() που ήδη εξετάσαμε, ενώ με την κλήση της μεθόδου show() ζητάμε να εμφανίζονται όλα τα συστατικά της εφαρμογής που έχουμε δηλώσει ότι θέλουμε να εμφανίζονται στην οθόνη του χρήστη.

## ii. Η κλάση GensettingsPanel

Η κλάση GensettingsPanel που θα εξετάσουμε τώρα, είναι η κλάση της οποίας αντικείμενα είναι όλα τα πλαίσια ρυθμίσεων ταλαντωτών. Οι πρώτες γραμμές κώδικα της συγκεκριμένης κλάσης φαίνονται παρακάτω.

```
package gr.teicrete.audiosynth;

import javax.swing.*;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
import java.awt.Color;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.text.NumberFormatter;
```

Στις παραπάνω γραμμές κώδικα βλέπουμε και πάλι τη δήλωση του πακέτου στην οποία περιλαμβάνεται αυτή η κλάση και τις δηλώσεις import. Ενώ όλες οι κλάσεις εισάγονται μια προς μια παρατηρούμε ότι η δήλωση ‘‘import javax.swing.*;’’ εισάγει ολόκληρο το πακέτο javax.swing. Στην ουσία δεν εισάγει ολόκληρο το πακέτο απλά το κάνει διαθέσιμο στον προγραμματιστή. Οι κλάσεις που θα εισαχθούν είναι αυτές που θα

δηλωθούν αργότερα στον κώδικα. Το πακέτο swing περιέχει τα βασικά γραφικά συστατικά μιας εφαρμογής.

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

    protected double amplitude;// μεταβλητή για το πλάτος
    protected double frequency; //μεταβλητή για την συχνότητα
    private double index; //μεταβλητή για τον δείκτη διαμόρφωσης
    protected double indexMultiplied; // μεταβλητή που θα ισούται είτε με index*amplitude είτε με
                    //index*frequency
    private int modulatorType = 2; //μεταβλητή που δηλώνει αν ο ταλαντωτής είναι διαμορφωτής
                // και τι είδους
    private double maximumfrequency=10000.0; //Η μέγιστη συχνότητα ενός ταλαντωτή

    JLabel gentypelabel; // η ετικέτα ενός ταλαντωτή
    JSlider ampGenSlider = new JSlider(JSlider.HORIZONTAL, 0, 200, 0 );//slider πλάτους
    JLabel ampsliderLabel = new JLabel("Amplitude ");//ετικέτα slider πλάτους
    JLabel ampsliderunit = new JLabel("dB");//ετικέτα “dB”
    JLabel maxfreq = new JLabel("Maximum Frequency  (Hz)"); //ετικέτα “Maximum Frequency (Hz)”
    JSlider frGenSlider = new JSlider(JSlider.HORIZONTAL, 0, 200, 0 );//slider συχνότητας
    JLabel  frsliderLabel = new JLabel("Frequency ");//ετικέτα “Frequency”
    JLabel  frsliderunit = new JLabel("Hz"); //ετικέτα “Hz”
    JSlider indexslider = new JSlider(JSlider.HORIZONTAL,0,1000,0); // slider δείκτη διαμόρφωσης
    JLabel  indexlabel = new JLabel("Index"); // ετικέτα ”Index”
    java.text.NumberFormat numberFormat = java.text.NumberFormat.getInstance() ;// καθορισμός της μορφής
    NumberFormatter formatter = new NumberFormatter(numberFormat);    //των αριθμών των πεδίων
                                        //τιμών
    JFormattedTextField genAmpTextField = new JFormattedTextField(formatter); // πεδίο τιμών πλάτους
    JFormattedTextField genFrTextField = new JFormattedTextField(formatter); //πεδίο τιμών συχνότητας
    JFormattedTextField genInTextField = new JFormattedTextField(formatter); //πεδίο τιμών δείκτη διαμόρφ.
    JRadioButton [] checkbuttons = new JRadioButton[4]; //checkboxes
    JButton [] testbuttons = new JButton[10];
```

Στις παραπάνω γραμμές κώδικα βλέπουμε την δήλωση της κλάσης και το μπλοκ αρχικοποίησης. Βλέπουμε καταρχήν ότι και αυτή η κλάση έχει δηλωθεί public που όπως είπαμε σημαίνει ότι είναι «ορατή» από οποιαδήποτε άλλη κλάση μέσα στο ίδιο πακέτο. Η κλάση αυτή επεκτείνει την κλάση JPanel για την οποία έχουμε ήδη μιλήσει. Τέλος, βλέπουμε ότι υλοποιεί τις διασυνδέσεις ChangeListener και ActionListener για τις οποίες θα μιλήσουμε παρακάτω. Το μπλοκ αρχικοποίησης δεν θα σχολιαστεί αλλά ο αναγνώστης μπορεί να πάρει στοιχεία για αυτό από τα σχόλια που είναι ενσωματωμένα στον κώδικα.

Παρακάτω φαίνονται οι πρώτες γραμμές της μεθόδου δημιουργού της κλάσης.

```
public GensettingsPanel (String gentype){
    super();
    this.setBackground(AudioSynth.mygreen);
    this.setBorder(BorderFactory.createMatteBorder(0,0,1,0,Color.BLACK));
    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints constraints = new GridBagConstraints();
    this.setLayout(gridbag);
    gentypelabel = new JLabel(gentype);
    ampGenSlider.setFocusable(false);
```

```
ampGenSlider.setBackground(AudioSynth.mygreen);
frGenSlider.setFocusable(false);
frGenSlider.setBackground(AudioSynth.mygreen);
indexslider.setFocusable(false);
indexslider.setBackground(AudioSynth.mygreen);
genAmpTextField.setValue(new Float( 10*(Math.log(amplitude/32767))));
genFrTextField.setValue(new Float( frequency));
genInTextField.setValue(new Float( 0));
ButtonGroup group = new ButtonGroup();
for (int i=0; i<4; i++){
checkbuttons[i] = new JRadioButton();
checkbuttons[i].setFocusable(false);
checkbuttons[i].addActionListener(this);
checkbuttons[i].setBackground(AudioSynth.mygreen);
group.add(checkbuttons[i]);
}
```

Παρατηρούμε ότι η μέθοδος δημιουργός έχει σαν όρισμα ένα String. Πράγματι, στην κλάση AudioSynth είδαμε ότι δημιουργήσαμε αντικείμενα της κλάσης GensettingsPanel, με δηλώσεις όπως αυτή: `GensettingsPanel sinesettingspanel_01 = new GensettingsPanel("SINE (1,2)");`. Το όρισμα αυτό θα χρησιμοποιηθεί στην γραμμή 7 του constructor ώστε να θέσει αυτό το String στην ετικέτα του πλαισίου ρυθμίσεων ταλαντωτή με την δήλωση: `gentypelabel = new JLabel(gentype);`.

Κατά τα άλλα, το παραπάνω μπλοκ κώδικα φροντίζει για κάποια στοιχεία των γραφικών συστατικών όπως το χρώμα και οι αρχικές τιμές που θα έχουν τα ρυθμιστικά και τα πεδία κειμένου.

```
//amplitude slider
AudioSynth.buildConstraints(constraints, 1, 1, 4, 1, 0,100 );
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(ampGenSlider, constraints);
ampGenSlider.addChangeListener(this);
this.add(ampGenSlider);
//frequency slider
AudioSynth.buildConstraints(constraints, 1, 2, 4, 1, 0,100 );
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(frGenSlider, constraints);
frGenSlider.addChangeListener(this);
this.add(frGenSlider);
//index slider
AudioSynth.buildConstraints(constraints, 1, 3, 4, 1, 0,100 );
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(indexslider, constraints);
indexslider.addChangeListener(this);
this.add(indexslider);
}
```

Στο παραπάνω μπλοκ κώδικα φαίνεται ηπροσθήκη κάποιων συστατικών στο πάνελ και πάλι με τη χρήση της μεθόδου setConstraints της κλάσης AudioSynth. Για λόγους οικονομίας δεν έχει συμπεριληφθεί παραπάνω η προσθήκη όλων των συστατικών του πλαισίου ρυθμίσεων ταλαντωτή.

Παρατηρούμε ότι για κάθε slider του πλαισίου καλείται η μέθοδος addChangeListener(this). Επίσης, αν ανατρέξουμε σε προηγούμενες γραμμές κώδικα θα δούμε ότι για κάθε κουμπί επιλογής μέγιστης συχνότητα ταλαντωτή, καλείται η μέθοδος addActionListener(this). Οι δύο αυτές μέθοδοι σχετίζονται με τις διασυνδέσεις που υλοποιεί η συγκεκριμένη κλάση και με τις μεθόδους που θα αναλυθούν παρακάτω.

```
public void stateChanged (ChangeEvent evt) {
Object src = evt.getSource();
      if ((src== ampGenSlider)&&(!AudioSynth.isMovingonPlaypanel)) {
        amplitude = (double)(ampGenSlider.getValue())/(double)(200);
        genAmpTextField.setValue(new Float(10*(Math.log(((float)(ampGenSlider.getValue())/200)))));
      }
      else if ((src==frGenSlider)&&(!AudioSynth.isMovingonPlaypanel)) {
                        frequency=(this.maximumfrequency)*(double)(frGenSlider.getValue())
                                          /(double)(frGenSlider.getMaximum());
      genFrTextField.setValue(new  Float((float)(frequency)));
      }

      else if(src == indexslider){
              if(this.modulatorType==0){
            index=((double)(indexslider.getValue())/(double)(indexslider.getMaximum()));
                              genInTextField.setValue(newFloat((float)((double)(indexslider.getValue())/
                              (double)(indexslider.getMaximum()))));

      }
      else if(this.modulatorType==1){
            index=((double)(20*indexslider.getValue())/(double)(indexslider.getMaximum()));
             genInTextField.setValue(new Float((float)((double)(20*indexslider.getValue())
                                                /(double)(indexslider.getMaximum()))));
      }
      else{
             index=((double)(indexslider.getValue())/(double)(indexslider.getMaximum()));
            genInTextField.setValue(newFloat((float)((double)(indexslider.getValue())
                  /(double)(indexslider.getMaximum()))));
      }
sendAmplitudeFrequencyIndex();
}
```

Η παραπάνω μέθοδος καλείται όποτε αλλάζουν οι ρυθμίσεις σε κάποιο από τα sliders. Αυτό συμβαίνει ως αποτέλεσμα της κλήσης για αυτά τα συστατικά της μεθόδου addChangeListener(this). Μέσα σε αυτή τη μέθοδο ελέγχουμε καταρχάς ως αποτέλεσμα ποίου slider κλήθηκε η μέθοδος και αναλόγως θέτουμε τις τιμές των μεταβλητών amplitude, frequency και index καθώς και τις τιμές που εμφανίζονται στα αντίστοιχα πεδία τιμών. Για την περίπτωση που το slider που κάλεσε την μέθοδο είναι αυτό του δείκτη διαμόρφωσης, ελέγχεται αν οι ρυθμίσεις αφορούν σε AM ή FM διαμορφωτή αφού στις δύο αυτές περιπτώσεις ο δείκτης διαμόρφωσης έχει διαφορετικό εύρος τιμών. Ο έλεγχος αυτός γίνεται με τη χρήση της integer μεταβλητής modulatorType η οποία όπως

θα δούμε παίρνει την τιμή ‘0’ όταν οι ρυθμίσεις αφορούν σε AM διαμορφωτή και την τιμή ‘1’ όταν αφορά σε FM διαμορφωτή.

Παρακάτω φαίνεται η μέθοδος actionPerformed η οποία καλείται όποτε αλλάζει η επιλογή που αφορά στη μέγιστη συχνότητα ταλαντωτή.

```
public void actionPerformed (ActionEvent evt) {
      Object src = evt.getSource();
if(checkbuttons[0].isSelected()){
      this.maximumfrequency=20.0;
frGenSlider.setValue((int)((double)(frequency)*(double)(frGenSlider.getMaximum())/maximumfrequency));
}
else if(src == checkbuttons[1]){
     this.maximumfrequency=200.0;
frGenSlider.setValue((int)((double)(frequency*(double)(frGenSlider.getMaximum()))/maximumfrequency));
}
     else if(src == checkbuttons[2]){
     this.maximumfrequency=2000.0;
frGenSlider.setValue((int)((double)(frequency*(double)(frGenSlider.getMaximum()))/maximumfrequency));
}
     else {
     this.maximumfrequency=10000.0;
frGenSlider.setValue((int)((double)(frequency*(double)(frGenSlider.getMaximum()))/maximumfrequency));
     }
    if ( frequency>this.maximumfrequency){
    frequency=maximumfrequency;
    this.sendAmplitudeFrequencyIndex();
    }
}
```

Η μέθοδος αυτή αλλάζει την τιμή της μεταβλητής maximumfrequency. Η μεταβλητή αυτή σε συνδυασμό με την θέση του slider της συχνότητας καθορίζει την συχνότητα του ταλαντωτή. Η μέθοδος αυτή αποτελεί την υλοποίηση του interface ActionListener.

Αν παρατηρήσουμε τις δύο παραπάνω μεθόδους θα δούμε ότι μέσα στον κώδικα και των δύο καλείται η μέθοδος sendAmplitudeFrequencyIndex(). Η μέθοδος αυτής φαίνεται παρακάτω.

```
private void sendAmplitudeFrequencyIndex(){
      switch (modulatorType){
      case 0:
          indexMultiplied = index*amplitude ;
           AudioSynth.getInstance().setAmplitudeFrequencyandIndex(amplitude,frequency,index*amplitude);
           break;
      case 1:
          indexMultiplied = index*frequency ;
           AudioSynth.getInstance().setAmplitudeFrequencyandIndex(amplitude,frequency,index*frequency);
           break;
      case 2:
          indexMultiplied = index ;
          AudioSynth.getInstance().setAmplitudeFrequencyandIndex(amplitude,frequency,index);
          break;
     }
```

```
    }
```

Μέσα από τη μέθοδο αυτή καλούμε τη μέθοδο setAmplitudeFrequencyandIndex της κλάσης AudioSynth που έχουμε ήδη εξετάσει. Η μέθοδος αυτή «στέλνει» ουσιαστικά τις ρυθμίσεις για το πλάτος, τη συχνότητα και το δείκτη διαμόρφωσης για κάθε ταλαντωτή στην κλάση AudioSynth ώστε αυτή να τις χειριστεί καταλλήλως. Παρατηρούμε ότι για την περίπτωση του δείκτη διαμόρφωσης δεν στέλνεται ακριβώς αυτή η τιμή αλλά το γινόμενο του δείκτη διαμόρφωσης με το πλάτος ή τι συχνότητα του ταλαντωτή, ανάλογα με το τι είδους διαμορφωτής είναι αυτός. Με τη χρήση και πάλι της μεταβλητής modulatorType ελέγχεται αν ο ταλαντωτής είναι διαμορφωτής και τι είδους. Στη συνέχεια θα δούμε πως παίρνει τις πιθανές τιμές της αυτή η μεταβλητή. Για αυτό τον λόγο θα εξετάσουμε τη μέθοδο setModulatorPanel που φαίνεται παρακάτω:

```
    protected void setModulatorPanel(int integer){
          switch (integer){
          case 0:
               this.modulatorType=0;//AM Modulator
               this.index=((double)(indexslider.getValue())/(double)(indexslider.getMaximum()));
               this.indexMultiplied = index*amplitude ;
               this.genInTextField.setValue(new Float((float)(index)));
               this.indexlabel.setText("AM Index");
               break;
         case 1:
               this.modulatorType=1;//FM modulator
this.index=((double)(20*indexslider.getValue())/(double)(indexslider.getMaximum()));
               this.indexMultiplied = index*frequency ;
               this.genInTextField.setValue(new Float((float)(index)));
               this.indexlabel.setText("FM Index");
               break;
         case 2:
               this.modulatorType=2;//carrier
               this.indexlabel.setText("Index");
               break;
          }
    }
```

Παρατηρούμε ότι η μέθοδος αυτή έχει δηλωθεί ως “protected” που σημαίνει ότι μπορούμε να την καλέσουμε μέσα από άλλες κλάσεις του ίδιου πακέτου. Πράγματι, καλούμε αυτήν τη μέθοδο όποτε αλλάζουν οι ρυθμίσεις του πλέγματος ταλαντωτών ή του πλαισίου τύπων διαμόρφωσης και επομένως μέσα από τις κλάσεις SwitchPanel και ModSettingsPanel.

Η μέθοδος αυτή ορίζει τον τρόπο που θα λειτουργεί το πλαίσιο ρυθμίσεων ταλαντωτή καθώς και την εμφάνιση του ανάλογα με το αν αφορά σε διαμορφωτή και τι τύπου. Το όρισμα “integer” της μεθόδου χρησιμοποιείται για να ενημερώσει την κλάση σε τι είδους ταλαντωτή απευθύνεται. Παρατηρούμε ότι η μεταβλητή modulatorType παίρνει κάθε φορά τις τιμές αυτού του ορίσματος και είναι 0 όταν αφορά σε διαμορφωτή πλάτους, 1 όταν αφορά σε διαμορφωτή συχνότητας και 2 όταν αφορά σε ταλαντωτή – φορέα.

Η τελευταία μέθοδος της κλάσης GensettingsPanel φαίνεται παρακάτω.

```
protected double getMaximumFrequency(){
      return this.maximumfrequency;
}
```

Η μέθοδος αυτή, όπως δηλώνει και το όνομα της, έχει σκοπό να ενημερώνει τις άλλες κλάσεις για την τιμή της μέγιστης συχνότητας ταλαντωτή. Συγκεκριμένα, καλείται από την κλάση AudioSynth ώστε αυτή να γνωρίζει την τιμή της μέγιστης συχνότητας και αναλόγως αυτή η τιμή να ισχύει και για την μέγιστη συχνότητα που μπορεί να δωθεί σε έναν ταλαντωτή από την κίνηση του κέρσορα στο Mouseboard.

## iii. Η κλάση ModSettingsPanel

Η κλάση ModSettingsPanel όπως έχουμε πει, είναι η κλάση στην οποία ορίζεται η μορφή του πλαισίου επιλογής τύπων διαμόρφωσης. Οι πρώτες γραμμές του κώδικα της συγκεκριμένης κλάσης φαίνονται παρακάτω:

```
package gr.teicrete.audiosynth;

import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JRadioButton;
import javax.swing.ImageIcon;
import javax.swing.BorderFactory;
import java.awt.Color;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
```

Σε αυτές τις γραμμές κώδικα βλέπουμε την δήλωση του πακέτου στο οποίο ανήκει η κλάση και τις δηλώσεις ‘‘import’’ που απαιτούνται για τον ορισμό της κλάσης. Παρακάτω βλέπουμε τη δήλωση της κλάσης και το μπλοκ αρχικοποίησης.

```
public class ModSettingsPanel extends JPanel implements ActionListener {
      JRadioButton [] check = new JRadioButton[2];//πίνακας που περιέχει τα δύο κουμπιά
      JLabel titlelabel = new JLabel("Modulation Settings");//ετικέτα πλαισίου
      JLabel firsttypelabel = new JLabel("1st Modulation type:");ετικέτα πρώτου κουμπιού
      JLabel sectypelabel = new JLabel("2nd Modulation type:"); ετικέτα δεύτερου κουμπιού
//δημιουργία αντικειμένου ImageIcon με τη χρή ση του εικονιδίου AM.gif
      ImageIcon    AMicon   = new ImageIcon("C:/AudioSynth/src/gr/teicrete/audiosynth/AM.gif");
//δημιουργία αντικειμένου ImageIcon με τη χρή ση του εικονιδίου FM.gif
      ImageIcon    FMicon   = new ImageIcon("C:/AudioSynth/src/gr/teicrete/audiosynth/FM.gif");
```

Όπως ήταν αναμενόμενο και αυτή η κλάση επεκτείνει την κλάση JPanel του πακέτου javax.swing. Παρατηρούμε ότι υλοποιείται η διασύνδεση ActionListener. Επεξηγήσεις για το μπλοκ αρχικοποίησης δίνονται από τα σχόλια που συνοδεύουν τον κώδικα.

Παρακάτω βλέπουμε τις πρώτες γραμμές της μεθόδου – δημιουργού της κλάσης.

```
public  ModSettingsPanel(){
super();
this.setBorder(BorderFactory.createMatteBorder(0,0,1,0,Color.BLACK));
GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
this.setLayout(gridbag);
for (int i =0; i<2; i++){
        check[i] = new JRadioButton(AMicon);
        check[i].setSelectedIcon(FMicon);
        check[i].addActionListener(this);
       }
```

Στην πρώτη γραμμή της μεθόδου δημιουργού έχουμε το κάλεσμα στην μέθοδο δημιουργό της «υπερκλάσης» δηλαδή της κλάσης JPanel ( super();). Χρωματίζονται μαύρα τα όρια του πλαισίου και γίνονται οι απαραίτητες ενέργειες για την δημιουργία και την υιοθέτηση ενός διαχειριστή διάταξης για τα διάφορα γραφικά συστατικά. Μέσα στο βρόχο "for" δημιουργούνται τα δύο κουμπιά που είναι αντικείμενα της κλάσης JRadioButton και γίνονται οι απαραίτητες ρυθμίσεις για την αρχική τους κατάσταση. Επίσης καλείται η μέθοδος "addActionListener(this);" ώστε αυτά τα κουμπιά να σχετίζονται με την διασύνδεση ActionListener.

Ο διαχειριστής διάταξης που χρησιμοποιείται και για αυτήν την κλάση είναι ο διαχειριστής δομημένων πλεγμάτων (GridBagLayout). Η προσθήκη των συστατικών στις επιθυμητές θέσεις γίνεται με τρόπο παρόμοιο με αυτόν των προηγούμενων κλάσεων. Ενδεικτικά βλέπουμε την προσθήκη δύο μόνο συστατικών.

```
//Title
        AudioSynth.buildConstraints(constraints, 0, 0, 6, 1, 100, 100);
        constraints.fill = GridBagConstraints.CENTER;
        constraints.anchor = GridBagConstraints.CENTER;
        gridbag.setConstraints(titlelabel, constraints);
         titlelabel.setSize(50,50);
        this.add(titlelabel);
.....
.....
    //2nd Modulation  RadioButton
        AudioSynth.buildConstraints(constraints, 4, 1, 1, 1, 100, 100);
        constraints.fill = GridBagConstraints.NONE;
        constraints.anchor = GridBagConstraints.WEST;
        gridbag.setConstraints(check[1], constraints);
        this.add(check[1]);

   }
```

Η κλάση ModSettingsPanel περιέχει μόνο μία μέθοδο και αυτή είναι η μέθοδος actionPerformed() ο ρόλος της οποίας είναι η υλοποίηση της μοναδικής διασύνδεσης της κλάσης. Στις παρακάτω γραμμές βλέπουμε αυτή την μέθοδο:

```
//////////////////////////ActionListener Implementation//////////////////////
    public void actionPerformed (ActionEvent evt) {
        Object actionsrc = evt.getSource();
```

```
    if(actionsrc ==check[0]&&!check[0].isSelected()){
          AudioSynth.MODULATION1TYPE=0;
    }
    else if(actionsrc ==check[0]&& check[0].isSelected()){
          AudioSynth.MODULATION1TYPE=1;
    }
    else if(actionsrc ==check[1]&&!check[1].isSelected()){
          AudioSynth.MODULATION2TYPE=0;
    }
    else if(actionsrc ==check[1]&& check[1].isSelected()){
         AudioSynth.MODULATION2TYPE=1;
    }
        AudioSynth.getInstance().updateOscillatorsSettings();
  }
```

Η παραπάνω μέθοδος καλείται όποτε αλλάζει κάποιος από τους δύο τύπους διαμόρφωσης. Όπως είναι κατανοητό, στη μέθοδο αυτή πραγματοποιούνται διαφορετικές ενέργειες ανάλογα με το ποιο από τα δύο κουμπιά πατήθηκε και ανάλογα με την κατάσταση στην οποία τέθηκε, δηλαδή AM ή FM. Σύμφωνα με αυτές τις συνθήκες, κάθε φορά αλλάζουν οι τιμές των μεταβλητών MODULATION1TYPE και MODULATION2TYPE της κλάσης AudioSynth. Αυτές οι δύο μεταβλητές ενημερώνουν τις υπόλοιπες κλάσεις για την κατάσταση του πρώτου και του δεύτερου διακόπτη επιλογής τύπων διαμόρφωσης αντίστοιχα. Όταν έχουν την τιμή ‘0’ ο αντίστοιχος τύπος διαμόρφωσης είναι AM, ενώ όταν έχουν την τιμή ‘1’ ο αντίστοιχος τύπος διαμόρφωσης είναι FM.

## iv. Η κλάση SwitchPanel

Παρακάτω βλέπουμε τις πρώτες γραμμές κώδικα της κλάσης SwitchPanel ο ορισμός της οποίας γίνεται στο αρχείο SwitchPanel.java. Σε αυτές τις γραμμές βλέπουμε τη δήλωση του πακέτου στο οποίο ανήκει η κλάση και τις δηλώσεις “import”.

```
package gr.teicrete.audiosynth;

import javax.swing.JPanel;
import javax.swing.ImageIcon;
import javax.swing.BorderFactory;
import javax.swing.JRadioButton;
import java.awt.Insets;
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.MouseListener;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.ActionEvent;
```

Παρακάτω βλέπουμε τη δήλωση της κλάσης SwitchPanel και το μπλοκ αρχικοποίησης αυτής.

```
public class Switchpanel  extends JPanel implements MouseListener,ActionListener {
```

```
    ImageIcon   switchicon   = new ImageIcon("C:/AudioSynth/src/gr/teicrete/audiosynth/grid.gif");
    ImageIcon   switchselectedicon   = new ImageIcon("C:/AudioSynth/src/gr/teicrete/audiosynth/gridselected.gif");
//Δημιουργία δύο αντικειμένων της κλάσης ImageIcon με χρήση δύο gif αρχείων
    JRadioButton switchbutton = new JRadioButton(switchicon);//δημιουργία ενός διακόπτη του grid
    JRadioButton [] switches = new JRadioButton [9];// δημιουργία πίνακα  που θα αποθηκεύει τους διακόπτες
                                                    // του grid
    JRadioButton switchoutlined;//μεταβλητή στην οποία «αποθηκεύεται» ο επιλεγμένος διακόπτης
    Insets insets = new Insets(0,0,0,0);
```

Παρατηρούμε ότι και αυτή η κλάση επεκτείνει την κλάση JPanel του πακέτου Swing, ενώ υλοποιούνται οι διασυνδέσεις ActionListener και MouseListener.

Το παρακάτω μπλοκ κώδικα περιλαμβάνει ολόκληρη τη μέθοδο – δημιουργό της κλάσης.

```
public Switchpanel () {
super();
this.setBorder(BorderFactory.createMatteBorder(1,1,1,1,Color.BLACK));
GridLayout thisgrid = new GridLayout (3 , 3 , 0 , 0);
this.setLayout(thisgrid);
for (int i=0; i<9; i++) {
      switches[i] = new JRadioButton(switchicon);
      switches[i].setSelectedIcon(switchselectedicon);
      switches[i].addMouseListener(this);
      switches[i].addActionListener(this);
      switches[i].setBackground(AudioSynth.mygreen);
      switches[i].setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY,3));
      this.add(switches[i]);
}
switchoutlined = switches[0];
switchoutlined.setBorderPainted(true);
}
```

Στις πρώτες γραμμές της μεθόδου, έχουμε την κλήση της μεθόδου δημιουργού της υπερκλάσης, τη δημιουργία ενός μαύρου περιγράμματος του πλαισίου και τις απαραίτητες ρυθμίσεις για τη διάταξη των συστατικών στο πλαίσιο. Παρατηρούμε ότι ο διαχειριστής διάταξης είναι αντικείμενο της κλάσης GridLayout και όχι GridBagLayout όπως ίσχυε για τις προηγούμενες κλάσεις. Η διάταξη GridLayout είναι διάταξη πλέγματος όπου σε αντίθεση με τη διάταξη GridBagLayout κάθε συστατικό μπορεί να καταλαμβάνει μόνο ένα από τα κελιά του πλέγματος.

Στο βρόχο "for" που ακολουθεί έχουμε τη δημιουργία όλων των διακοπτών του πλέγματος, τις απαραίτητες ρυθμίσεις για αυτούς και τέλος την πρόσθεση τους στο πλαίσιο. Στις ρυθμίσεις για αυτούς τους διακόπτες περιλαμβάνεται ο ορισμός του εικονιδίου που θα εμφανίζεται σε κάθε διακόπτη όταν κάνουμε κλικ επάνω του, η συσχέτιση κάθε διακόπτη με τις δύο διασυνδέσεις που υλοποιεί η κλάση καθώς και η δημιουργία του γκρι πλαισίου γύρω από τους διακόπτες. Η δημιουργία του πλαισίου γίνεται με την γραμμή κώδικα: switches[i].setBorder (BorderFactory.createLineBorder (Color.LIGHT_GRAY,3)); Βλέπουμε ότι δημιουργούμε το πλαίσιο γύρω από κάθε διακόπτη παρόλο που μόνο ένας από αυτούς είναι κάθε φορά επιλεγμένος. Πράγματι, για να είναι ορατό το παραπάνω πλαίσιο δεν αρκεί η παραπάνω γραμμή κώδικα, αλλά είναι απαραίτητες και οι δύο τελευταίες γραμμές κώδικα της μεθόδου δημιουργού. Σε αυτές

τις γραμμές ορίζεται ποιος διακόπτης είναι αρχικά επιλεγμένος (switchoutlined = switches[0];) και στη συνέχεια ζητείται αυτός ο διακόπτης να εμφανίσει το ανάλογο πλαίσιο (switchoutlined.setBorderPainted(true);).

Παρακάτω βλέπουμε τέσσερις «κενές» μεθόδους της κλάσης.

```
public void mouseReleased (MouseEvent evt) {
}
public void mousePressed (MouseEvent evt) {
}
public void mouseExited (MouseEvent evt) {
}
public void mouseEntered (MouseEvent evt) {
}
```

Αυτές οι μέθοδοι καλούνται σε διάφορες ενέργειες του χρήστη σχετικές με τους διακόπτες του πλέγματος, αλλά φυσικά δεν συμβαίνει τίποτα. Οι μέθοδοι αυτοί υπάρχουν στον κώδικα μας γιατί μαζί με την μέθοδο που ακολουθεί είναι απαραίτητες για την υλοποίηση της διασύνδεσης MouseListener.

```
public void mouseClicked (MouseEvent evt) {
      Object mousesrc = evt.getSource();
if (evt.getButton()==3){
      if (mousesrc == switches[0]) {
         switchoutlined.setBorderPainted(false);
         switches[0].setBorderPainted(true);
         switchoutlined=switches[0];
         AudioSynth.VISIBLEGSPANEL=0;
         AudioSynth.getInstance().showGensettingsPanel();
……..
      if (mousesrc == switches[8]){
         AudioSynth.VISIBLEGSPANEL=8;
         switchoutlined.setBorderPainted(false);
         switches[8].setBorderPainted(true);
         switchoutlined=switches[8];
         AudioSynth.getInstance().showGensettingsPanel();
      }
}}
```

Η μέθοδος καλείται όποτε ο χρήστης κάνει κλικ σε κάποιον από τους διακόπτες. Όλος ο κώδικας αυτής της μεθόδου περιέχεται σε μια " if " συνθήκη έτσι ώστε η μέθοδος αυτή να αφορά μόνο στις ενέργειες που πρέπει να γίνουν όταν ο χρήστης κάνει δεξί κλικ σε κάποιον από τους διακόπτες. Όπως είναι αναμενόμενο από την περιγραφή του AudioSynth στο προηγούμενο κεφάλαιο , σε αυτή τη μέθοδο ορίζεται ποίου διακόπτη το περίγραμμα να εμφανίζεται γκρι καθώς και ποιο πλαίσιο ρυθμίσεων ταλαντωτή να εμφανίζεται στην οθόνη του χρήστη. Για λόγους οικονομίας χώρου, παραπάνω δεν εμφανίζεται το σώμα της μεθόδου που αφορά σε όλους τους διακόπτες του πλέγματος αλλά μόνο σε δύο από αυτούς.

Παρακάτω βλέπουμε τις πρώτες γραμμές τις μεθόδου actionPerformed που όπως έχουμε δει και κατά τη μελέτη των προηγούμενων κλάσεων χρησιμοποιείται για την υλοποίηση της διασύνδεσης ActionListener.

```
public void actionPerformed (ActionEvent evt) {

Object actionsrc = evt.getSource();
if((actionsrc==switches[0])&&(switches[0].isSelected()==true)){
      AudioSynth.Osc1ON = 1;
}
if((actionsrc==switches[0])&&(switches[0].isSelected()==false)){
      AudioSynth.Osc1ON = 0;
}
.....
.....
```

Η μέθοδος αυτή καλείται όποτε ο χρήστης κάνει αριστερό κλικ σε κάποιον από τους διακόπτες του πλέγματος. Παραπάνω, βλέπουμε μόνο τον κώδικα που αφορά σε έναν από τους διακόπτες του πλέγματος. Όπως βλέπουμε ανάλογα με το αν θέτουμε σε κατάσταση ON ή σε κατάσταση OFF τον εν λόγω διακόπτη η μεταβλητή OSC1ON της κλάσης AudioSynth παίρνει την τιμή 1 ή 0. Στις επόμενες γραμμές τις μεθόδου ορίζονται ακριβώς τα ίδια και για τους υπόλοιπους διακόπτες του πλέγματος ταλαντωτών και για τις ανάλογες μεταβλητές(OSC20N , OSC3ON, OSC4ON…..).

Παρακάτω βλέπουμε τον κώδικα που ολοκληρώνει τη μέθοδο ActionPerformed.

```
//////////////////Change GRIDSTATE and Update Panels////////////////
if(AudioSynth.Osc4ON==0&&AudioSynth.Osc5ON==0&&AudioSynth.Osc6ON==0&&AudioSynth.Osc7ON==0&&
AudioSynth.Osc8ON==0&&AudioSynth.Osc9ON==0){
      AudioSynth.GRIDSTATE=100;
}
else
if((AudioSynth.Osc1ON==0&&AudioSynth.Osc2ON==0&&AudioSynth.Osc3ON==0)&&(AudioSynth.Osc4ON==1 ||
AudioSynth.Osc5ON==1 || AudioSynth.Osc6ON ==1 )&&(AudioSynth.Osc7ON==0 && AudioSynth.Osc8ON==0 &&
AudioSynth.Osc9ON ==0)){
      AudioSynth.GRIDSTATE=10;
      }
else
if((AudioSynth.Osc4ON==1||AudioSynth.Osc5ON==1||AudioSynth.Osc6ON==1)&&(AudioSynth.Osc7ON==0&&AudioS
ynth.Osc8ON==0&&AudioSynth.Osc9ON==0)){
      AudioSynth.GRIDSTATE=110;
}
.....
}
```

Στις παραπάνω γραμμές κώδικα, ελέγχονται ανά ομάδες οι τιμές των μεταβλητών που καθορίσαμε παραπάνω και αναλόγως δίνονται τιμές στην μεταβλητή ακεραίων GRIDSTATE της κλάσης AudioSynth. Οι τιμές αυτής της μεταβλητής χρησιμοποιούνται από την κλάση GridDecoder προκειμένου να αναγνωρίζεται τι είδους «κύκλωμα» σύνθεσης ήχου πρέπει να υλοποιηθεί. Παραπάνω δίνονται ανά περίσταση τρεις από τις πιθανές τιμές που μπορεί να πάρει η μεταβλητή GRIDSTATE. Οι καταστάσεις του πλέγματος ταλαντωτών που περιγράφει αυτή η μεταβλητή αφορούν στο αν και σε ποιες

γραμμές του πλέγματος υπάρχει τουλάχιστον 1 ταλαντωτής ο οποίος είναι σε κατάσταση ON. Το ποιες τιμές αντιστοιχούν σε ποιες καταστάσεις του grid θα το δούμε κατά την εξέταση της κλάσης GridDecoder που όπως έχουμε ήδη δει φροντίζει για την «αποκωδικοποίηση» του πλέγματος ταλαντωτών.

## v. Η κλάση GeneralSettingsPanel

Παρακάτω βλέπουμε τις πρώτες γραμμές κώδικα της κλάσης GeneralSettingsPanel η οποία ορίζεται μέσα στο αρχείο GeneralSettingsPanel.java. Σε αυτές τις γραμμές κώδικα βλέπουμε τη δήλωση του πακέτου στο οποίο ανήκει η κλάση και τις δηλώσεις ‘‘import’’.

```
package gr.teicrete.audiosynth;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.Color;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import java.awt.GridBagLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.text.NumberFormatter;
```

Όπως και σε όλες τις κλάσεις που εξετάσαμε μέχρι τώρα τις παραπάνω γραμμές κώδικα ακολουθούν η δήλωση της κλάσης και το μπλοκ αρχικοποίησης αυτής.

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

	private JLabel audiolabel = new JLabel("Audio:");//ετικέτα “Audio”
	private JButton showmouse = new JButton("MOUSEBOARD");// κουμπί “MOUSEBOARD”
	private JRadioButton on = new JRadioButton("ON");// επιλογή “ON”
	private JRadioButton off = new JRadioButton("OFF"); // επιλογη “OFF”
	private JSlider gamplitudeslider = new JSlider(JSlider.HORIZONTAL, 0, 32767, 0 );// slider πλάτους
	private JLabel  sliderlabel = new JLabel("Global Amplitude");// ετικέτα "Global Amplitude"
	private JButton reset = new JButton("!");// κουμπί reset
	private double globalAmplitude; //μεταβλητή αποθήκευσης συνολικού πλάτους
//////////////Ρυθμίσεις για τις τιμές που εμφανίζονται στα ανάλογα πεδία//////////////////////////////////
	private java.text.NumberFormat numberFormat = java.text.NumberFormat.getInstance() ;
		private NumberFormatter formatter = new NumberFormatter(numberFormat);
		private JFormattedTextField gAmpTextField = new JFormattedTextField(formatter);
	private JFormattedTextField gPeakTextField = new JFormattedTextField(formatter);
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
	private Insets insets = new Insets (0,0,0,0);//περιθώρια για χρήση στον ορισμό της μορφής κουμπιών
	private ButtonGroup bgroup = new ButtonGroup();// ομάδα επιλογών(ON/OFF)
	private double MaxOut = 0;// μεταβλητή αποθήκευσης μέγιστης τιμής πλάτους
```

Από τη δήλωση της κλάσης, βλέπουμε ότι και αυτή η κλάση επεκτείνει την κλάση JPanel, ενώ υλοποιεί και τις διασυνδέσεις ActionListener και ChangeListener.

Παρακάτω βλέπουμε το πρώτο μέρος της μεθόδου δημιουργού της κλάσης.

```
	public GeneralSettingsPanel() {
	 super();
```

```
this.setBorder(BorderFactory.createMatteBorder(1,0,0,0,Color.BLACK));
reset.setForeground(Color.darkGray);
reset.setFocusable(false);
reset.setMargin(insets);
GridBagLayout ggridbag = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
this.setLayout(ggridbag);

bgroup.add(on);
bgroup.add(off);

off.setSelected(true);
gamplitudeslider.setValue(gamplitudeslider.getMaximum());
globalAmplitude = gamplitudeslider.getValue();
```

Οι πρώτες γραμμές της μεθόδου είναι παρόμοιες με τις αντίστοιχες των προηγούμενων κλάσεων, αφού και πάλι έχουμε κλήση στη μέθοδο – δημιουργό της υπερκλάσης, το σχηματισμό του περιγράμματος του γραφικού αντικειμένου που ορίζει αυτή η κλάση και ρυθμίσεις για την εμφάνιση και την τοποθέτηση των συστατικών στο πλαίσιο γενικών ρυθμίσεων. Παρατηρούμε ότι με τις δηλώσεις `bgroup.add(on);` και `bgroup.add(off);` καθορίζουμε ότι μόνο μία από τις επιλογές on και off μπορεί να είναι κάθε φορά επιλεγμένη. Στη συνέχεια ορίζουμε ότι αρχικά επιλεγμένη είναι η κατάσταση off, ότι το slider αρχικά του συνολικού πλάτους είναι στη μέγιστη του θέση και δίνουμε στη μεταβλητή globalAmplitude την τιμή που αντιστοιχεί σε αυτήν την θέση του slider.

Οι επόμενες γραμμές της μεθόδου αφορούν στην πρόσθεση των συστατικών στο πλαίσιο με διάταξη που ορίζεται από την κλάση GridBagLayout. Παρόμοιες διαδικασίες έχουμε δει μέχρι τώρα στον ορισμό αρκετών κλάσεων και για αυτό τον λόγο δεν παρουσιάζεται αυτό το μπλοκ κώδικα.

Παρακάτω, βλέπουμε την μέθοδο η οποία υλοποιεί τη διασύνδεση ChangeListener. Όπως έχουμε δει και κατά την εξέταση των προηγούμενων κλάσεων η μέθοδος αυτή ονομάζεται “stateChanged”.

```
public void stateChanged (ChangeEvent evt) {
globalAmplitude = gamplitudeslider.getValue();
this.gAmpTextField.setValue(new
Float(10*(Math.log((globalAmplitude/(double)(gamplitudeslider.getMaximum()))))));
}
```

Η μέθοδος αυτή καλείται, όποτε μετακινείται το slider που ελέγχει το συνολικό πλάτος. Έτσι, βλέπουμε ότι στην πρώτη γραμμή κώδικα της μεθόδου ορίζεται η μεταβλητή globalAmplitude να αποθηκεύει κάθε φορά την τιμή που αντιστοιχεί στη θέση του slider.Η εντολή της μεθόδου μετατρέπει αυτήν την τιμή που μπορεί να είναι από 0 ως 1 σε dB και την εμφανίζει στο πεδίο που βρίσκεται αριστερά από το slider.

Στις παρακάτω γραμμές κώδικα βλέπουμε την μέθοδο actionPerformed η οποία υλοποιεί τη διασύνδεση ActionListener της κλάσης.

```
public void actionPerformed (ActionEvent evt) {
Object actionsrc = evt.getSource();
```

```
if (actionsrc.equals(on)) {
        AudioSynth.playing = true;
        AudioSynth.getInstance().play();
} else if (actionsrc.equals(off)) {
        AudioSynth.getInstance().g.line.drain();
        AudioSynth.playing  = false;
} else if (actionsrc.equals(showmouse)) {
        AudioSynth.getInstance().mouseboard.setVisible(true);
}
else if (actionsrc.equals(reset)) {
        this.MaxOut=0;
        gPeakTextField.setValue(new Float(0.0f));
        reset.setForeground(Color.darkGray);
}
}
```

Η μέθοδος αυτή καλείται όταν ο χρήστης κάνει κλικ στις επιλογές ON και OFF , στο κουμπί mouseboard ή στο κουμπί reset (!).

Από τον κώδικα ο οποίος αφορά στην πρώτη συνθήκη "if" βλέπουμε τι γίνεται όταν ο χρήστης θέσει το AudioSynth σε κατάσταση "ON". Σε αυτήν την περίπτωση η Boolean μεταβλητή "playing" της κλάσης AudioSynth γίνεται αληθής και καλείται η μέθοδος play της ίδιας κλάσης. Αυτά τα δύο βήματα αποτελούν απαραίτητες συνθήκες για να παράγει ήχο η εφαρμογή AudioSynth. Μπορούμε να θυμηθούμε, ότι μέσα στη μέθοδο play δημιουργείται το thread μέσα στο οποίο γίνεται ότιδήποτε έχει σχέση με τη σύνθεση και την αναπαραγωγή του ήχου.

Όταν η μέθοδος καλείται εξαιτίας του ότι ο χρήστης έθεσε την εφαρμογή σε κατάσταση "OFF" συμβαίνουν οι ενέργειες που συνδέονται με την δεύτερη συνθήκη "if". Καταρχήν βλέπουμε ότι όπως είναι λογικό, η μεταβλητή "playing" που εξετάσαμε πριν λίγο παίρνει την τιμή "false". Η δήλωση AudioSynth.getInstance().g.line.drain(); δίνει εντολή να αναπαραχθούν όσα bytes ήχου είχαν αποθηκευτεί σε ένα buffer που αποθηκεύεται ο προς αναπαραγωγή ήχος. Θα αναφερθούμε με περισσότερες λεπτομέρειες σε αυτό το buffer, όταν θα εξετάσουμε την κλάση AudioSetup.

Όταν η μέθοδος καλείται εξαιτίας του κουμπιού "MOUSEBOARD", μέσω της εντολής AudioSynth.getInstance().mouseboard.setVisible(true); το αντίστοιχο παράθυρο εμφανίζεται στην οθόνη του χρήστη.

Τέλος, όταν η μέθοδος καλείται εξαιτίας του κουμπιού reset, το θαυμαστικό που αποτελεί την ετικέτα αυτού του κουμπιού γίνεται γκρι και το πεδίο που εμφανίζει την τιμή του πλάτους εξόδου παίρνει στιγμιαία την τιμή 0. Επίσης η μεταβλητή "MaxOut" παίρνει την τιμή 0. Την χρήση αυτής της μεταβλητής θα εξετάσουμε τώρα κατά την εξέταση της μεθόδου "setPeak". Η μέθοδος αυτή φαίνεται παρακάτω.

```
protected void setPeak(double AudioOut){
if(AudioOut > MaxOut){
      MaxOut=AudioOut;
 this.gPeakTextField.setValue(new Float( 10*(Math.log(MaxOut))));
```

```
}
if(MaxOut>=1){
      reset.setForeground(Color.red);
}
}
```

Η μέθοδος αυτή δέχεται ως όρισμα μια double μεταβλητή "AudioOut". Αυτή η μέθοδος καλείται μέσα από την κλάση AudioSetup και το όρισμα που δέχεται αποτελεί τη στιγμιαία τιμή του πλάτους εξόδου. Έτσι με την πρώτη συνθήκη "if", αν η τιμή του ορίσματος είναι μεγαλύτερη από την τιμή της μεταβλητής "MaxOut", η μεταβλητή αυτή παίρνει την τιμή του ορίσματος. Στη συνέχεια η νέα τιμή της μεταβλητής "MaxOut" εμφανίζεται στο πεδίο εμφάνισης τιμών του πλάτους εξόδου. Η μεταβλητή "MaxOut" αποθηκεύει δηλαδή, τη μεγαλύτερη στιγμιαία τιμή του πλάτους εξόδου. Για να αλλάξει αυτή η τιμή πρέπει να κληθεί και πάλι αυτ'η η μέθοδος αλλά με μια μεγαλύτερη τιμή ορίσματος. Η επόμενη συνθήκη "if" ορίζει ότι όταν η μεταβλητή "MaxOut" πάρει τιμή μεγαλύτερη από το 1 που αντιστοιχεί στα 0 dB, το θαυμαστικό του κουμπιού reset παίρνει κόκκινο χρώμα.

Παρακάτω φαίνεται η τελευταία μέθοδος της κλάσης GeneralSettingsPanel.

```
protected double getGlobalAmplitude(){
return globalAmplitude;
}
```

Η μέθοδος αυτή καλείται επίσης από την κλάση AudioSetup και επιστρέφει την τιμή της μεταβλητής globalAmplitude, που όπως έχουμε δει αντιπροσωπεύει την ενδεχόμενη υποβάθμιση που προκαλούμε στο σήμα εξόδου.

## vi. Η κλάση MouseBoard

Σε αυτό το σημείο, έχουμε εξετάσει όλες τις κλάσεις που σχετίζονται με το κύριο παράθυρο της εφαρμογής AudioSynth. Η κλάση MouseBoard, ο ορισμός της οποίας γίνεται στο αρχείο MouseBoard.java είναι η τελευταία κλάση γραφικών που θα εξετάσουμε και φυσικά αντικείμενο της οποίας αποτελεί το ανάλογο παράθυρο. Παρακάτω βλέπουμε την δήλωση του πακέτου στο οποίο ανήκει η κλάση και όλες τις δηλώσεις "import" για τη συγκεκριμένη κλάση.

```
package gr.teicrete.audiosynth;

import java.awt.Color;
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.JPanel;
```

Παρακάτω βλέπουμε τη δήλωση της κλάσης και το μπλοκ αρχικοποίησης αυτής.

```
public class MouseBoard  implements MouseMotionListener,MouseListener {
```

```
private JFrame mouseboardframe  = new JFrame("Mouseboard");
private JPanel playpanel = new JPanel();
```

Παρατηρούμε ότι η κλάση δεν επεκτείνει την κλάση JFrame που όπως έχουμε δει αφορά στα συνηθισμένα παράθυρα. Αντί αυτού βλέπουμε ότι ένα παράθυρο δημιουργείται μέσα στο μπλοκ αρχικοποίησης. Φυσικά δημιουργείται και το αντικείμενο JPanel που σχετίζεται με αυτό το παράθυρο. Τέλος, παρατηρούμε ότι η κλάση MouseBoard υλοποιεί τις διασυνδέσεις MouseMotionListener και MouseListener.

Στις επόμενες γραμμές βλέπουμε τη μέθοδο – δημιουργό της κλάσης.

```
public MouseBoard() {
mouseboardframe.setBounds (400,0,600,600);
mouseboardframe.setContentPane (playpanel);
playpanel.addMouseMotionListener(this);
playpanel.addMouseListener(this);
mouseboardframe.setDefaultCloseOperation (JFrame.HIDE_ON_CLOSE);
mouseboardframe.setVisible(false);
}
```

Στην πρώτη γραμμή της μεθόδου ορίζεται η αρχική τοποθεσία και το αρχικό μήκος και πλάτος του παραθύρου. Στη δεύτερη γραμμή συσχετίζουμε το παράθυρο αυτό με το αντικείμενο της κλάσης JPanel που έχουμε ήδη δημιουργήσει. Στις επόμενες δύο γραμμές συσχετίζουμε αυτό το αντικείμενο με τις διασυνδέσεις που υλοποιεί η κλάση μας. Τέλος, στην προτελευταία γραμμή κώδικα ορίζουμε ότι όταν κλείνουμε το παράθυρο αυτό απλά να μην είναι ορατό στο χρήστη, ενώ στην τελευταία γραμμή ορίζουμε ότι αρχικά το παράθυρο δεν εμφανίζεται.

Στη συνέχεια, βλέπουμε δύο μεθόδους που υλοποιούν τη διασύνδεση MouseListener.

```
public void mouseEntered (MouseEvent evt) {
      mouseboardframe.requestFocus();
}
public void mouseExited (MouseEvent evt) {

      AudioSynth.isMovingonPlaypanel=false;
       AudioSynth.getInstance().requestFocus();

}
```

Η πρώτη μέθοδος, όπως άλλωστε δηλώνει και το όνομα της, περιέχει τις ενέργειες που γίνονται όταν ο κέρσορας μπαίνει εντός του JPanel του παραθύρου. Η εντολή ‘‘mouseboardframe.requestFocus();’’ ορίζει ‘ότι σε αυτήν την περίπτωση το παράθυρο Mouseboard πρέπει να είναι επιλεγμένο.

Αντίστοιχα, η δεύτερη μέθοδος καλείται όταν ο κέρσορας φεύγει από το “playpanel”. Σε αυτήν την περίπτωση το παράθυρο Mouseboard παύει να είναι επιλεγμένο και επιλέγεται το κύριο παράθυρο της εφαρμογής.

Όπως έχουμε ήδη αναφέρει, για να είναι δυνατή η υλοποίηση της συγκεκριμένης διασύνδεσης πρέπει να υπάρχουν και άλλες τρεις μέθοδοι έστω και αν αυτές είναι κενές. Αυτές οι μέθοδοι φαίνονται παρακάτω.

```
public void mouseReleased (MouseEvent evt) {
}
public void mousePressed (MouseEvent evt) {
}
public void mouseClicked (MouseEvent evt) {
}
```

Στη συνέχεια, θα εξετάσουμε την υλοποίηση της διασύνδεσης MouseMotionListener. Για την υλοποίηση της συγκεκριμένης διασύνδεσης είναι απαραίτητος ο ορισμός δύο μεθόδων, η πρώτη από τις οποίες φαίνεται παρακάτω.

```
public void mouseMoved(MouseEvent e) {
Object mousesrc = e.getSource();
if(!mousesrc.equals(playpanel)) {
      return;
}
AudioSynth.isMovingonPlaypanel=true;
float mouseX = e.getX();
float mouseY = e.getY();
float playpanelWidth = playpanel.getWidth()-1 ;
float playpanelHeight = playpanel.getHeight()-1;
float xRatio = mouseX/playpanelWidth;
float yRatio = mouseY/playpanelHeight;
AudioSynth.getInstance().updateSliderPosition(xRatio, yRatio);
}
```

Αυτή η μέθοδος καλείται όποτε ο κέρσορας κινείται μέσα στο "playpanel". Η πρώτη συνθήκη "if" ελέγχει ότι πράγματι αυτό ήταν που οδήγησε στην κλήση της μεθόδου. Στην πραγματικότητα αυτό το μπλοκ κώδικα θα μπορούσε να λείπει, αφού έχουμε καλέσει τη μέθοδο "addMouseMotionListener" μόνο για το συστατικό "playpanel" και έτσι είναι αδύνατο να καλέσουμε τη μέθοδο διαφορετικά.

Στη συνέχεια ορίζουμε ότι η μεταβλητή "isMovingonPlaypanel" είναι "true". Όπως έχουμε δει αυτή η μεταβλητή ορίζεται στην κλάση AudioSynth και χρησιμοποιείται από την κλάση GensettingsPanel προκειμένου αυτή να γνωρίζει αν η κίνηση των slider του αντίστοιχου πλαισίου είναι αποτέλεσμα της κίνησης του κέρσορα στο Mouseboard ή όχι. Στις επόμενες τέσσερις τιμές δημιουργούνται οι μεταβλητές που αποθηκεύουν τη θέση του κέρσορα στον οριζόντιο και κατακόρυφο άξονα του MouseBoard καθώς και οι μεταβλητές που αποθηκεύουν το μεταβλητό μήκος και ύψος του παραθύρου. Με τη χρήση αυτών των μεταβλητών δημιουργούμε τους λόγους "xRatio" και "yRatio", οι οποίοι στην τελευταία γραμμή κώδικα της μεθόδου αποτελούν τα ορίσματα για την κλήση στην μέθοδο "updateSliderPosition" της κλάσης AudioSynth που έχουμε ήδη εξετάσει.

Η δεύτερη απαραίτητη μέθοδος για την υλοποίηση της διασύνδεσης MouseMotionListener, φαίνεται παρακάτω.

```
public void mouseDragged(MouseEvent e){}
```

Η μέθοδος αυτή όπως παρατηρούμε είναι «κενή» και αυτό πρακτικά σημαίνει ότι όταν κάνουμε drag με τον κέρσορα μέσα στο "playpanel" δεν αλλάζει τίποτα στον ήχο που αναπαράγεται.

## vii. Η κλάση SineGenerator

Η κλάση SineGenerator, ο ορισμός της οποίας γίνεται στο αρχείο SineGenerator.java, είναι η πρώτη κλάση που θα εξετάσουμε και αφορά στη σύνθεση ήχου. Όπως ήδη έχουμε δει όταν κάναμε μια γενική περιγραφή των κλάσεων, αντικείμενα αυτής της κλάσης αποτελούν οι εννιά ημιτονικοί ταλαντωτές του AudioSynth. Αυτή η κλάση όπως και όλες οι υπόλοιπες που αφορούν στη σύνθεση και αναπαραγωγή ήχου θα εξηγηθούν όσο το δυνατόν πιο λεπτομερειακά.

Παρακάτω φαίνεται η δήλωση του πακέτου στο οποίο ανήκει η κλάση και η μοναδική δήλωση "import" της κλάσης.

```
package gr.teicrete.audiosynth;

import javax.sound.sampled.AudioFormat;
```

Με τον παραπάνω κώδικα, εισάγεται στην συγκεκριμένη κλάση η κλάση AudioFormat του πακέτου javax.sound.sampled. Οτιδήποτε αφορά στον ήχο στην java, βρίσκεται στο πακέτο javax.sound. Αυτό το πακέτο χωρίζεται στο πακέτο javax.sound.sampled το οποίο αφορά την παραγωγή ήχου μέσω δειγματοληψίας και το πακέτο javax.sound.midi το οποίο αφορά στην παραγωγή ήχου μέσω midi. Η εφαρμογή AudioSynth κάνει χρήση μόνο του πακέτου javax.sound.sampled.

Παρακάτω βλέπουμε τη γραμμή δήλωσης της κλάσης και το μπλοκ αρχικοποίησης αυτής.

```
public class SineGenerator {

        private double sinvalue;
    private int c;
    AudioFormat audioFormat = new AudioFormat (AudioFormat.Encoding.PCM_SIGNED ,44100 , 16 , 1 , 2 , 44100
, false);
```

Στις πρώτες γραμμές του μπλοκ αρχικοποίησης βλέπουμε ότι δημιουργούμε μια double μεταβλητή κινητής υποδιαστολής ("sinvalue") που θα χρησιμοποιηθεί όπως θα δούμε αργότερα για να αποθηκεύει την τιμή εξόδου του εκάστοτε ταλαντωτή. Η επόμενη μεταβλητή που δημιουργούμε είναι μια μεταβλητή ακεραίων που θα χρησιμοποιηθεί για τον υπολογισμό της θέσης διαβάσματος από ένα buffer.

Τέλος, στο μπλοκ αρχικοποίησης δημιουργούμε ένα αντικείμενο της κλάσης AudioFormat. Όπως δηλώνει και το όνομα αυτής της κλάσης, αντικείμενα που αποτελούν υποστάσεις αυτής περιγράφουν διάφορα format ήχου. Παρατηρούμε ότι η μέθοδος–δημιουργός αυτής της κλάσης δέχεται 7 ορίσματα. Το πρώτο όρισμα αφορά στην τεχνική κωδικοποίησης Audio. Στην συγκεκριμένη περίπτωση έχουμε επιλέξει PCM (Pulse Code Modulation) προσημασμένη. Το δεύτερο όρισμα αφορά στη συχνότητα δειγματοληψίας που έχει οριστεί να είναι 44100 samples/sec.. Το τρίτο όρισμα δηλώνει το μέγεθος ενός δείγματος σε bits. Στην συγκεκριμένη περίπτωση έχει οριστεί να είναι 16(2 bytes). Αυτό αποτελεί το λεγόμενο «βάθος bit». Στο επόμενο όρισμα δηλώνουμε τον αριθμό των audio καναλιών που επιθυμούμε. Σε αυτό το όρισμα έχει δοθεί η τιμή 1 που σημαίνει ότι το AudioSynth είναι μονοφωνικό. Το πέμπτο όρισμα που έχει την τιμή 2 αντιπροσωπεύει τον αριθμό των bytes σε κάθε frame ηχητικής πληροφορίας. Στο έκτο όρισμα δίνουμε την τιμή του "framerate", δηλαδή των audio frames που θα αναπαράγονται ανά δευτερόλεπτο. Στο format που έχουμε επιλέξει, το "framerate" συμπίπτει με τη συχνότητα δειγματοληψίας. Η δήλωση "false" στο τελευταίο όρισμα, δηλώνει ότι τα δεδομένα σε ένα sample δεν αποθηκεύονται σε σειρά big-Endian και επομένως αποθηκεύονται σε σειρά little-Endian .

Παρακάτω βλέπουμε τη μέθοδο – δημιουργό της κλάσης.

```
protected SineGenerator() {
}
```

Όπως βλέπουμε όταν καλούμε αυτή τη μέθοδο δεν συμβαίνει τίποτα αφού αυτή η μέθοδος είναι τελείως κενή. Καλούμε αυτή τη μέθοδο για να έχουμε αναφορά σε διαφορετικούς ταλαντωτές, όλη η λειτουργικότητα των οποίων περικλείεται στην παρακάτω μέθοδο.

```
protected  double generate (double amp, double freq, float[] table){
int readstep =(int) (Math.round(freq*(AudioSynth.sinetable.length/audioFormat.getSampleRate())));
if ( c < 0) {
      c=table.length-c;
}

while (c>=table.length){
       c=c%table.length;
}
sinvalue = amp *table[c];
c=c+readstep;
return sinvalue;
}
```

Όταν αυτή η μέθοδος καλείται επανειλημμένα, ο ταλαντωτής που περιγράφει αυτή η κλάση θα «διαβάζει» διάφορες τιμές από έναν πίνακα. Ο πίνακας αυτός υποδεικνύεται από το τελευταίο όρισμα της μεθόδου. Όπως είναι αναμενόμενο στην εφαρμογή AudioSynth αυτός ο πίνακας θα είναι πάντα ένας πίνακας στις θέσεις του οποίου είναι αποθηκευμένες οι τιμές μιας περιόδου μιας ημιτονικής κυματομορφής.

Στην πρώτη γραμμή της μεθόδου βλέπουμε την δημιουργία μιας μεταβλητής ακεραίων με το όνομα “readstep”. Η μεταβλητή αυτή θα αποτελεί το βήμα διαβάσματος για την σάρωση του πίνακα “table”.

Ο ταλαντωτής που θέλουμε να κατασκευάσουμε θα έχει σταθερή συχνότητα δειγματοληψίας και η συχνότητα που θέλουμε να παράγουμε θα υπολογίζεται μεταβάλλοντας κατάλληλα το βήμα διαβάσματος. Για ταλαντωτές τέτοιου είδους, ισχύει:

συχνότητα εξόδου = συχνότητα ταλαντωτή x βήμα διαβάσματος
↔ βήμα διαβάσματος = συχνότητα εξόδου / συχνότητα ταλαντωτή

Η συχνότητα εξόδου, δηλαδή η συχνότητα στην οποία θέλουμε να συντονίσουμε τον ταλαντωτή μας δίνεται από το όρισμα “freq” της μεθόδου. Η συχνότητα ταλαντωτή αποτελεί τη συχνότητα την οποία αναπαράγει ο ταλαντωτής αν διαβάζει όλες τις θέσεις του πίνακα ή αλλιώς αν έχει βήμα διαβάσματος ‘1’. Για τη συχνότητα αυτή ισχύει:

συχνότητα ταλαντωτή = συχνότητα δειγματοληψίας / θέσεις πίνακα

Επομένως η παραπάνω σχέση γίνεται:

βήμα διαβάσματος = συχνότητα εξόδου x (θέσεις πίνακα / συχνότητα δειγματοληψίας)

Αυτή ακριβώς η σχέση σε κώδικα Java εκφράζεται ως:

```
int readstep =(int) (Math.round(freq*(AudioSynth.sinetable.length/audioFormat.getSampleRate())));
```

Όλες οι πράξεις αποτελούν το όρισμα στη μέθοδο “round” της κλάσης Math. Η μέθοδος αυτή όταν δέχεται ένα όρισμα “double”, όπως πράγματι είναι το αποτέλεσμα της παραπάνω πράξης, το μετατρέπει σε “long int”. Ο λόγος που θέλουμε να συμβαίνει κάτι τέτοιο μπορεί να εξηγηθεί αν δούμε την παρακάτω σχέση:

Επόμενη θέση διαβάσματος = προηγούμενη θέση + βήμα διαβάσματος.

Αφού κάθε θέση του πίνακα έχει σαν δείκτη έναν ακέραιο αριθμό δεν μπορεί παρά και το βήμα διαβάσματος να είναι ακέραιος αριθμός.

Υπάρχουν τρεις τρόποι για τη μετατροπή ενός αριθμού κινητής υποδιαστολής σε ακέραιο. Η πρώτη είναι η μέθοδος της αποκοπής, στην οποία αποκόπτεται τελείως το δεκαδικό μέρος του αριθμού. Η δεύτερη είναι η μέθοδος της στρογγυλοποίησης την οποία πετυχαίνουμε με την μέθοδο “Math.round”. Κατά αυτή τη μέθοδο εξάγεται η τιμή της θέσης με την κοντινότερη ακέραια τιμή. Η μέθοδος αυτή είναι πιο δαπανηρή υπολογιστικά από την μέθοδο της αποκοπής αλλά και πιο ακριβής . Η πιο ακριβής είναι η μέθοδος της ζυγισμένης προσέγγισης η οποία όμως δεν προτιμήθηκε επειδή θα κατανάλωνε σημαντική υπολογιστική ισχύ. Σε αυτή τη μέθοδο η τιμή εξόδου θα υπολογιζόταν σαν ποσοστό επί των τιμών των δύο κοντινότερων ακέραιων θέσεων με βάση το δεκαδικό μέρος.

Φυσικά και με τις τρεις παραπάνω μεθόδους δεν θα πετυχαίναμε την απόλυτη ακρίβεια. Αυτές τις απώλειες τις περιορίζουμε αισθητά έχοντας έναν πολύ μεγάλο πίνακα στον οποίο αποθηκεύουμε τις τιμές μιας ημιτονικής κυματομορφής. Όπως ήδη έχουμε δει ο πίνακας της κλάσης AudioSynth, "sinetable", έχει 1.048.576 θέσεις.

Η μέθοδος "generate" έχει δηλωθεί ότι επιστρέφει μια "double" τιμή. Αυτή είναι η τιμή που αποθηκεύεται στην μεταβλητή sinvalue. Μέσα στη μέθοδο δηλώνουμε ότι: sinvalue = amp*table[c]; . Δηλαδή, η έξοδος του ταλαντωτή κάθε στιγμή θα ισούται με το πλάτος του επί την τιμή που είναι αποθηκευμένη σε μία από τις θέσεις του πίνακα. Η θέση του πίνακα c είναι όπως έχουμε δει και στο μπλοκ αρχικοποίησης, ένας ακέραιος αριθμός με αρχική τιμή 0. Μέσα στη μέθοδο "generate" δηλώνουμε ότι: c=c+readstep; Έτσι, καθώς θα καλούμε επαναλαμβανόμενα τη μέθοδο η τιμή c θα ξεκινάει από το 0 και κάθε φορά θα αυξάνεται κατά το βήμα διαβάσματος. Με αυτόν τον τρόπο όμως, η τιμή της μεταβλητής c κάποια στιγμή θα υπερβεί τον αριθμό των θέσεων του πίνακα, πράγμα που φυσικά δεν είναι επιθυμητό αφού ο ρόλος της μεταβλητής c είναι να υποδεικνύει πια θέση πίνακα πρέπει κάθε φορά να διαβαστεί. Εξίσου μη αποδεκτό θα ήταν η μεταβλητή c να πάρει αρνητική τιμή. Για αυτόν τον λόγο υπάρχει μέσα στη μέθοδο η παρακάτω συνθήκη "if".

```
if ( c < 0) {
      c=table.length-c;
}
```

Με τις παραπάνω γραμμές κώδικα δίνουμε εντολή στον επεξεργαστή αν η τιμή της μεταβλητής c είναι μικρότερη από το 0 τότε να αποθηκευτεί σε αυτή η τιμή table.length – c που ισούται με το μήκος του πίνακα συν την απόλυτη τιμή της μεταβλητής c.

Με τον παραπάνω κώδικα δώσαμε στη μεταβλητή c μία θετική τιμή, αλλά δεν λύσαμε το πρόβλημα αφού αυτή η τιμή είναι σίγουρα μεγαλύτερη από τον αριθμό των θέσεων του πίνακα. Το πρόβλημα λύνεται οριστικά με τις παρακάτω γραμμές κώδικα.

```
while (c>=table.length){
      c=c%table.length;
}
```

Με αυτόν τον βρόχο "while" ζητάμε από τον επεξεργαστή να εκτελεί συνεχώς την πράξη c=c%table.length; αν η τιμή του c είναι μεγαλύτερη από το μήκος του πίνακα. Η πράξη αυτή επιστρέφει στην μεταβλητή c το υπόλοιπο της διαίρεσης της μεταβλητής αυτής με το μήκος του πίνακα. Είναι απαραίτητο, αυτή η πράξη να γίνεται επαναλαμβανόμενα γιατί μπορεί να υπάρξουν περιπτώσεις όπου το υπόλοιπο αυτής της διαίρεσης μπορεί να είναι επίσης μεγαλύτερο από το μήκος του πίνακα.

Με την παραπάνω συνθήκη "if" και τον βρόχο "while" καταφέρνουμε στη ουσία να διαβάζουμε κυκλικά έναν buffer, με όποιο βήμα διαβάσματος και αν αντιστοιχεί στη συχνότητα στην οποία θέλουμε να συντονίσουμε τον ταλαντωτή μας.

## viii. Η κλάση CircuitAssembler

Η κλάση CircuitAssembler είναι αυτή στην οποία ορίζονται οι διάφορες «πράξεις» της σύνθεσης ήχου. Συγκεκριμένα, ορίζεται η πρόσθεση δύο σημάτων και οι διαμορφώσεις πλάτους και συχνότητας ενός ταλαντωτή. Η κλάση αυτή ορίζεται στο αρχείο CircuitAssembler.java και παρακάτω βλέπουμε τις πρώτες γραμμές του κώδικα της.

```
package gr.teicrete.audiosynth;

public class CircuitAssembler {

        protected CircuitAssembler() {
        }
```

Στην πρώτη γραμμή κώδικα της κλάσης βλέπουμε και πάλι τη δήλωση του πακέτου στο οποίο ανήκει η κλάση. Παρατηρούμε ότι ο ορισμός της κλάσης δεν περιέχει μπλοκ αρχικοποίησης. Επίσης, βλέπουμε ότι η μέθοδος δημιουργός της κλάσης είναι «κενή». Αυτό πρακτικά σημαίνει ότι όταν δημιουργούμε ένα αντικείμενο αυτής της κλάσης το κάνουμε ώστε να έχουμε μια αναφορά για την χρήση των μεθόδων της κλάσης που θα εξετάσουμε παρακάτω.

```
    protected double addSignals (double sig1 , double sig2){
     double rtn;
     rtn=sig1+sig2;
     return rtn;
        }
```

Η παραπάνω μέθοδος είναι η μέθοδος με την οποία επιτυγχάνουμε την προσθετική σύνθεση. Παρατηρούμε ότι είναι μια πολύ απλή μέθοδος η οποία δέχεται δύο “double” ορίσματα και επιστρέφει επίσης μια “double” τιμή, η οποία αποτελεί το άθροισμα των δύο ορισμάτων. Για να έχει νόημα η κλήση αυτής της μεθόδου, πρέπει να είναι επαναλαμβανόμενη και τα ορίσματα να αποτελούν τις στιγμιαίες τιμές των δειγμάτων ακουστικών σημάτων.

Παρατηρούμε, ότι αυτή η μέθοδος προσθέτει μόνο δύο τιμές. Έχουμε όμως δει ότι είναι πιθανόν να θέλουμε να προσθέσουμε τρία σήματα, δηλαδή τις εξόδους τριών ταλαντωτών της ίδιας γραμμής του πλέγματος ταλαντωτών. Σε αυτήν την περίπτωση η κλήση στην παραπάνω μέθοδο θα ήταν κάπως έτσι:

```
double sum = addSignals(addSignals(out1,out2), out3)
```

όπου sum είναι το άθροισμα των τριών σημάτων και out1, out2 και out3, οι έξοδοι τριών ταλαντωτών. Βέβαια, η μέθοδος αυτή είναι τόσο απλή που αντί να την καλούμε, ίσως είναι προτιμότερο απλά να γράψουμε: `sum = out1+out2+out3`.

Η παρακάτω μέθοδος είναι η μέθοδος που καλούμε προκειμένου να επιτύχουμε τη διαμόρφωση πλάτους ή συχνότητας ενός ταλαντωτή.

```
protected double modulateGenerator (SineGenerator G1,double amplitude, double frequency, double modulatingsignal, float[] table, int modulationtype){
    double rtn=0;
    switch (modulationtype){
        case 0:  //AM
            rtn= G1.generate((amplitude*modulatingsignal) , frequency ,table) ;
            break;
        case 1:  //FM
            rtn= G1.generate(amplitude , (frequency + modulatingsignal) ,table );
            break;
    }
    return rtn;
   }
```

Η παραπάνω μέθοδος δέχεται έξι ορίσματα. Το πρώτο από αυτά είναι ένα αντικείμενο της κλάσης "SineGenerator" το οποίο αντιστοιχεί στον ταλαντωτή τον οποίο θέλουμε να διαμορφώσουμε. Τα δύο επόμενα "double" ορίσματα, αντιστοιχούν σε τιμές συχνότητας και πλάτους αυτού του ταλαντωτή. Το όρισμα "modulatingsignal" αντιστοιχεί στο σήμα διαμόρφωσης. Το πέμπτο όρισμα αντιστοιχεί στον πίνακα από τον οποίο θα διαβάζει τιμές ο ταλαντωτής, ενώ το τελευταίο όρισμα χρησιμεύει στον ορισμό του τύπου διαμόρφωσης που επιτυγχάνεται με αυτή τη μέθοδο.

Στην πρώτη γραμμή του κώδικα αρχικοποιείται η μεταβλητή "rtn" που όπως θα δούμε αποθηκεύει την τιμή που επιστρέφει αυτή η μέθοδος. Στη συνέχεια υπάρχει μια συνθήκη "switch" στην οποία εξετάζονται οι περιπτώσεις της AM και FM διαμόρφωσης, ανάλογα με την τιμή της μεταβλητής "modulationtype".

Όταν η μεταβλητή "modulationtype" έχει την τιμή '0', έχουμε διαμόρφωση πλάτους. Στην περίπτωση αυτή η μεταβλητή "rtn" παίρνει την τιμή που επιστρέφει η μέθοδος "generate" για τον ταλαντωτή που αντιστοιχεί στο πρώτο όρισμα της μεθόδου. Όπως έχουμε δει η μέθοδος "generate" δέχεται τρία ορίσματα. Το πρώτο αντιστοιχεί στο πλάτος, το δεύτερο αντιστοιχεί στη συχνότητα και το τρίτο στον πίνακα από τον οποίο θα «διαβάζει» τιμές ο ταλαντωτής. Σε αυτήν την περίπτωση στη θέση του ορίσματος του πλάτους, βάζουμε το όρισμα "amplitude" της μεθόδου πολλαπλασιασμένο με το όρισμα modulating signal. Στη θέση του δεύτερου ορίσματος βάζουμε το όρισμα "frequency" το οποίο θα ορίζει τη συχνότητα του ταλαντωτή. Τέλος στο τρίτο όρισμα βάζουμε τον πίνακα που είχαμε σαν πέμπτο όρισμα στη μέθοδο "modulateGenerator".

Όταν η μεταβλητή "modulationtype" έχει την τιμή '1', έχουμε διαμόρφωση συχνότητας του ταλαντωτή που αντιστοιχεί στο πρώτο όρισμα της μεθόδου. Έτσι καλούμε για αυτόν την μέθοδο "generate" βάζοντας στο όρισμα της συχνότητας το το άθροισμα των ορισμάτων "frequency" και "modulatingsignal" αυτής της μεθόδου. Στο όρισμα του πλάτους βάζουμε την τιμή της μεταβλητής "amplitude" ενώ σαν τρίτο όρισμα της μεθόδου "generate" βάζουμε και πάλι το πέμπτο όρισμα της μεθόδου "modulateGenerator".

Παρατηρούμε ότι και στις δύο παραπάνω περιπτώσεις δεν αναφερθήκαμε καθόλου στον δείκτη διαμόρφωσης. Αυτό συμβαίνει γιατί για προγραμματιστικούς λόγους, ο δείκτης διαμόρφωσης συνδέεται με τους ταλαντωτές – διαμορφωτές, ενώ η συγκεκριμένη μέθοδος αναφέρεται μόνο στους διαμορφωτές – φορείς. Τον δείκτη διαμόρφωσης θα τον χρησιμοποιούμε κατά τον υπολογισμό του ορίσματος "modulatingsignal" που προηγείται πάντα της κλήσης σε αυτή τη μέθοδο.

## ix. Η κλάση GridDecoder

Στην κλάση GridDecoder γίνεται όπως έχουμε δει η «αποκωδικοποίηση» του πλέγματος ταλαντωτών. Δηλαδή, το αντικείμενο αυτής της κλάσης ελέγχει την κατάσταση του grid και με τη χρήση των αντικειμένων των κλάσεων SineGenerator και CircuitAssembler παράγει τις αριθμητικές τιμές που αναπαριστούν τον ήχο. Παρακάτω βλέπουμε τις πρώτες γραμμές κώδικα της κλάσης.

```
package gr.teicrete.audiosynth;

public class GridDecoder {

 CircuitAssembler Ass1  = new CircuitAssembler();// το μοναδικό αντικείμενο της κλάσης CircuitAssembler
 private double out1; //έξοδος ταλαντωτή 1
 private double out2;// έξοδος ταλαντωτή 2
 private double out3;// έξοδος ταλαντωτή 3
 private double out123;// αθροισμα εξόδων ταλαντωτών 1, 2 και 3
 private double out4;//έξοδος ταλαντωτή 4
 private double out5; //έξοδος ταλαντωτή 5
 private double out6; //έξοδος ταλαντωτή 6
 private double out456; //αθροισμα εξόδων ταλαντωτών 4,5 και 6
 private double out7; //έξοδος ταλαντωτή 7
 private double out8; //έξοδος ταλαντωτή 8
 private double out9; //έξοδος ταλαντωτή 9
 private double out789; //αθροισμα εξόδων ταλαντωτών 7,8 και 9

 protected GridDecoder() {
 }
```

Στις παραπάνω γραμμές βλέπουμε τη δήλωση του πακέτου στο οποίο ανήκει η κλάση, τη δήλωση της κλάσης, το μπλοκ αρχικοποίησης και τη μέθοδο - δημιουργό της κλάσης. Στο μπλοκ αρχικοποίησης παρατηρούμε ότι σε αυτήν την κλάση δημιουργείται το μοναδικό αντικείμενο της κλάσης CircuitAssembler που εξετάσαμε προηγουμένως. Όπως φαίνεται παραπάνω, και σε αυτήν την κλάση η μέθοδος δημιουργός δεν περιέχει καθόλου κώδικα.

Όλη η ουσία της κλάσης GridDecoder βρίσκεται στη μέθοδο "deGrid" που ακολουθεί.

```
    protected double deGrid(){

    double rtn=0;
```

```
    switch(AudioSynth.GRIDSTATE){
        case 100:  rtn=……;
        case 10:   rtn=……;
        case 110:  rtn=……;
        case 1:    rtn=……;
        case 101:  rtn=……;
        case 111:  rtn=……;
        case 11:   rtn=……;
    }
    return rtn;
}
```

Όπως φαίνεται παραπάνω, ολόκληρος ο κώδικας της μεθόδου είναι μια συνθήκη "switch" στην οποία ανάλογα με τις διαφορετικές τιμές της μεταβλητής "GRIDSTATE" της κλάσης "AudioSynth" υπολογίζεται και η τιμή που θα επιστρέφει η μέθοδος. Έχουμε ήδη δει ότι η μεταβλητή "GRIDSTATE" αλλάζει ανάλογα με τις ρυθμίσεις στο πλέγμα ταλαντωτών. Στις παραπάνω γραμμές κώδικα φαίνεται η γενική μορφή της μεθόδου, αφού οι ενέργειες που γίνονται σε κάθε περίπτωση θα εξηγηθούν ξεχωριστά.

Παρακάτω, βλέπουμε τις ενέργειες που γίνονται για την περίπτωση που η μεταβλητή "GRIDSTATE" έχει την τιμή '100'. Αυτή η τιμή αντιστοιχεί στην περίπτωση που στο πλέγμα ταλαντωτών τουλάχιστον ένας από τους ταλαντωτές της πρώτης οριζόντιας γραμμής είναι ενεργοποιημένος και κανένας ταλαντωτής των υπολοίπων γραμμών του πλέγματος δεν είναι ενεργοποιημένος.

```
case 100:
        out1=AudioSynth.Osc1ON*AudioSynth.getInstance().sine1osc.generate
(AudioSynth.getInstance().osc1settings[0],AudioSynth.getInstance().osc1settings[1],AudioSynth.sinetable);
    out2=AudioSynth.Osc2ON*AudioSynth.getInstance().sine2osc.generate
(AudioSynth.getInstance().osc2settings[0],AudioSynth.getInstance().osc2settings[1],AudioSynth.sinetable);
    out3=AudioSynth.Osc3ON*AudioSynth.getInstance().sine3osc.generate
(AudioSynth.getInstance().osc3settings[0],AudioSynth.getInstance().osc3settings[1],AudioSynth.sinetable);
    out123=Ass1.addSignals(Ass1.addSignals(out1,out2),out3);
    rtn = out123;
    break;
```

Στις παραπάνω γραμμές κώδικα υπολογίζεται η έξοδος καθενός από τους τρεις ταλαντωτές της πρώτης γραμμής και στην συνέχεια υπολογίζεται το άθροισμα τους με την κλήση στη μέθοδο "addSignals" της κλάσης "CircuitAssembler". Η έξοδος κάθε ταλαντωτή υπολογίζεται σαν γινόμενο μιας μεταβλητής με την τιμή που επιστρέφει η μέθοδος "generate" για κάθε ταλαντωτή. Η μεταβλητή αυτή όπως έχουμε δει, εκφράζει αν είναι ενεργοποιημένος ο αντίστοιχος ταλαντωτής. Για παράδειγμα, η μεταβλητή "OSC1ON" είναι '1' όταν είναι ενεργοποιημένος ο ταλαντωτής ενώ στην αντίθετη περίπτωση είναι '0'. Είναι προφανές ότι όταν είναι '0' και το γινόμενο που περιγράφει την έξοδο του ταλαντωτή θα είναι '0'. Τα ορίσματα που αφορούν στο πλάτος και τη συχνότητα στη μέθοδο "generate" για κάθε τιμή είναι οι τιμές που «επιστρέφονται» από το πλαίσιο ταλαντωτών ή το "MouseBoard" και έχουν αποθηκευτεί όπως έχουμε δει σε θέσεις πινάκων τριών θέσεων της κλάσης AudioSynth. Το όρισμα του πίνακα από τον οποίο θα διαβάζει τιμές ο κάθε ταλαντωτής είναι φυσικά ο πίνακας "sinetable" που

περιέχει τις τιμές μιας ημιτονικής κυματομορφής, τον οποίο επίσης δημιουργήσαμε στην κλάση AudioSynth.

Παρακάτω, βλέπουμε τον κώδικα που αντιστοιχεί στις περιπτώσεις όπου η μεταβλητή "GRIDSTATE" έχει τις τιμές '10' και '1'. Η συνθήκη '10' αντιστοιχεί στην περίπτωση όπου μόνο ταλαντωτές της δεύτερης γραμμής είναι ενεργοποιημένοι ενώ η συνθήκη '1' αντιστοιχεί στην περίπτωση που είναι ενεργοποιημένοι μόνο ταλαντωτές της τρίτης γραμμής.

```
case 10:
        out4=AudioSynth.Osc4ON*AudioSynth.getInstance().sine4osc.generate
(AudioSynth.getInstance().osc4settings[0],AudioSynth.getInstance().osc4settings[1],AudioSynth.sinetable);
        out5=AudioSynth.Osc5ON*AudioSynth.getInstance().sine5osc.generate
(AudioSynth.getInstance().osc5settings[0],AudioSynth.getInstance().osc5settings[1],AudioSynth.sinetable);
out6=AudioSynth.Osc6ON*AudioSynth.getInstance().sine6osc.generate
(AudioSynth.getInstance().osc6settings[0],AudioSynth.getInstance().osc6settings[1],AudioSynth.sinetable);
    out456=Ass1.addSignals(Ass1.addSignals(out4,out5),out6);
    rtn = out456;
    break;
case 1:
        out7=AudioSynth.Osc7ON*AudioSynth.getInstance().sine7osc.generate
(AudioSynth.getInstance().osc7settings[0],AudioSynth.getInstance().osc7settings[1],AudioSynth.sinetable);
out8=AudioSynth.Osc8ON*AudioSynth.getInstance().sine8osc.generate
(AudioSynth.getInstance().osc8settings[0],AudioSynth.getInstance().osc8settings[1],AudioSynth.sinetable);
out9=AudioSynth.Osc9ON*AudioSynth.getInstance().sine9osc.generate
(AudioSynth.getInstance().osc9settings[0],AudioSynth.getInstance().osc9settings[1],AudioSynth.sinetable);
    out789=Ass1.addSignals(Ass1.addSignals(out7,out8),out9);
    rtn = out789;
    break;
```

Ο κώδικας των δύο παραπάνω περιπτώσεων, είναι παρόμοιος με αυτόν της περίπτωσης που η μεταβλητή "GRIDSTATE" έχει την τιμή '100' και για αυτό δεν θα εξεταστεί.

Παρακάτω βλέπουμε τον κώδικα που αντιστοιχεί στην περίπτωση που η μεταβλητή "GRIDSTATE" έχει την τιμή '110'. Σε αυτήν την περίπτωση είναι ενεργοποιημένος ένας τουλάχιστον ταλαντωτής της πρώτης γραμμής του grid και ένας τουλάχιστον της δεύτερης γραμμής, ενώ όλοι οι ταλαντωτές της τρίτης γραμμής του πλέγματος ταλαντωτών είναι απενεργοποιημένοι.

```
case 110:
        out1=AudioSynth.Osc1ON*AudioSynth.getInstance().sine1osc.generate
(AudioSynth.getInstance().osc1settings[2],AudioSynth.getInstance().osc1settings[1],AudioSynth.sinetable);
        out2=AudioSynth.Osc2ON*AudioSynth.getInstance().sine2osc.generate
(AudioSynth.getInstance().osc2settings[2],AudioSynth.getInstance().osc2settings[1],AudioSynth.sinetable);
        out3=AudioSynth.Osc3ON*AudioSynth.getInstance().sine3osc.generate
(AudioSynth.getInstance().osc3settings[2],AudioSynth.getInstance().osc3settings[1],AudioSynth.sinetable);
        out123 = out1 + out2 +out3;
        out4=AudioSynth.Osc4ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine4osc,
AudioSynth.getInstance().osc4settings[0],AudioSynth.getInstance().osc4settings[1],out123,AudioSynth.sinetable,
AudioSynth.MODULATION1TYPE);
```

```
        out5=AudioSynth.Osc5ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine5osc,
AudioSynth.getInstance().osc5settings[0],AudioSynth.getInstance().osc5settings[1],out123,AudioSynth.sinetable,
AudioSynth.MODULATION1TYPE);
        out6=AudioSynth.Osc6ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine6osc,
AudioSynth.getInstance().osc6settings[0],AudioSynth.getInstance().osc6settings[1],out123,AudioSynth.sinetable,
AudioSynth.MODULATION1TYPE);
    rtn = out4+out5+out6;
    break;
```

Σε αυτήν την περίπτωση υπολογίζονται καταρχήν με παρόμοιο τρόπο όπως και πριν οι έξοδοι των ταλαντωτών της πρώτης γραμμής του πλέγματος ταλαντωτών. Το άθροισμα αυτών των εξόδων θα χρησιμοποιηθεί αργότερα για την διαμόρφωση των ταλαντωτών της δεύτερης γραμμής. Η τιμή της μεταβλητής 'rtn' που επιστρέφει η μέθοδος αποτελεί το άθροισμα των εξόδων των ταλαντωτών της δεύτερης γραμμής.

Βλέπουμε ότι ο υπολογισμός των εξόδων της πρώτης γραμμής γίνεται με παρόμοιο τρόπο με αυτόν της περίπτωσης όπου αυτοί οι ταλαντωτές ήταν φορείς. Η μόνη διαφορά έγκειταί στο ότι στην κλήση της μεθόδου "generate" το όρισμα του πλάτους είναι η τρίτη θέση του πίνακα που αποθηκεύει τις ρυθμίσεις κάθε ταλαντωτή ενώ πριν ήταν η πρώτη. Αυτό συμβαίνει γιατί στην τρίτη θέση αυτού του πίνακα αποθηκεύονται οι τιμές που εμπεριέχουν και τον δείκτη διαμόρφωσης είτε πρόκειται για διαμόρφωση πλάτους είτε για διαμόρφωση συχνότητας. Στην συγκεκριμένη κατάσταση του πλέγματος ταλαντωτών έχουμε δει ότι οι ταλαντωτές της πρώτης γραμμής θα είναι διαμορφωτές των ταλαντωτών της δεύτερης γραμμής.

Όπως βλέπουμε παραπάνω, οι έξοδοι των ταλαντωτών της δεύτερης γραμμής υπολογίζονται μέσω κλήσης στη μέθοδο "modulateGenerator" της κλάσης CircuitAssembler. Όπως είδαμε κατά την εξέταση αυτής η κλάσης η μέθοδος "modulateGenerator" έχει ένα όρισμα το οποίο αντιστοιχεί στο σήμα διαμόρφωσης. Στη θέση αυτού του ορίσματος έχουμε εισάγει το άθροισμα των εξόδων των ταλαντωτών της πρώτης γραμμής. Παρατηρούμε επίσης ότι στο τελευταίο όρισμα της μεθόδου έχουμε εισάγει την τιμή της μεταβλητής MODULATION1TYPE της κλάσης AudioSynth. Αυτή η μεταβλητή καθορίζεται από το πλαίσιο επιλογής τύπων διαμόρφωσης και αφορά στον τύπο διαμόρφωσης που πραγματοποιούν οι ταλαντωτές τις πρώτης γραμμής είτε στους ταλαντωτές της δεύτερης γραμμής, είτε σε αυτούς της τρίτης γραμμής. Τα υπόλοιπα ορίσματα της μεθόδου τα οποία αφορούν στο πλάτος και στη συχνότητα των ταλαντωτών – φορέων, παίρνουνε τις τιμές που είναι αποθηκευμένες στους ανάλογους πίνακες τις κλάσης "AudioSynth".

Παρακάτω βλέπουμε τον κώδικα που αντιστοιχεί στις περιπτώσεις που η τιμή της μεταβλητής "GRIDSTATE" είναι '101' ή '11'. Αυτές οι τιμές «περιγράφουν» την περίπτωση που οι ταλαντωτές της πρώτης γραμμής διαμορφώνουν αυτούς της τρίτης και την περίπτωση που οι ταλαντωτές της δεύτερης γραμμής διαμορφώνουν τους ταλαντωτές της τρίτης γραμμής, αντίστοιχα.

```
case 101:
        out1=AudioSynth.Osc1ON*AudioSynth.getInstance().sine1osc.generate
(AudioSynth.getInstance().osc1settings[2],AudioSynth.getInstance().osc1settings[1],AudioSynth.sinetable);
```

```
        out2=AudioSynth.Osc2ON*AudioSynth.getInstance().sine2osc.generate
(AudioSynth.getInstance().osc2settings[2],AudioSynth.getInstance().osc2settings[1],AudioSynth.sinetable);
        out3=AudioSynth.Osc3ON*AudioSynth.getInstance().sine3osc.generate
(AudioSynth.getInstance().osc3settings[2],AudioSynth.getInstance().osc3settings[1],AudioSynth.sinetable);
         out123 = out1 + out2 +out3;
        out7=AudioSynth.Osc7ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine7osc,
AudioSynth.getInstance().osc7settings[0],AudioSynth.getInstance().osc7settings[1],out123,AudioSynth.sinetable,
AudioSynth.MODULATION1TYPE);
        out8=AudioSynth.Osc8ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine8osc,
AudioSynth.getInstance().osc8settings[0],AudioSynth.getInstance().osc8settings[1],out123,AudioSynth.sinetable,
AudioSynth.MODULATION1TYPE);
        out9=AudioSynth.Osc9ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine9osc,
AudioSynth.getInstance().osc9settings[0],AudioSynth.getInstance().osc9settings[1],out123,AudioSynth.sinetable,
AudioSynth.MODULATION1TYPE);
    rtn = out7+out8+out9;
    break;

case 11:
        out4=AudioSynth.Osc4ON*AudioSynth.getInstance().sine4osc.generate
(AudioSynth.getInstance().osc4settings[2],AudioSynth.getInstance().osc4settings[1],AudioSynth.sinetable);
        out5=AudioSynth.Osc5ON*AudioSynth.getInstance().sine5osc.generate
(AudioSynth.getInstance().osc5settings[2],AudioSynth.getInstance().osc5settings[1],AudioSynth.sinetable);
        out6=AudioSynth.Osc6ON*AudioSynth.getInstance().sine6osc.generate
(AudioSynth.getInstance().osc6settings[2],AudioSynth.getInstance().osc6settings[1],AudioSynth.sinetable);
    out456 = out4 + out5 +out6;
        out7=AudioSynth.Osc7ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine7osc,
AudioSynth.getInstance().osc7settings[0],AudioSynth.getInstance().osc7settings[1],out456,AudioSynth.sinetable,
AudioSynth.MODULATION2TYPE);
        out8=AudioSynth.Osc8ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine8osc,
AudioSynth.getInstance().osc8settings[0],AudioSynth.getInstance().osc8settings[1],out456,AudioSynth.sinetable,
AudioSynth.MODULATION2TYPE);
        out9=AudioSynth.Osc9ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine9osc,
AudioSynth.getInstance().osc9settings[0],AudioSynth.getInstance().osc9settings[1],out456,AudioSynth.sinetable,
AudioSynth.MODULATION2TYPE);
    rtn = out7+out8+out9;
    break;
```

Οι δύο παραπάνω περιπτώσεις είναι ανάλογες με αυτήν που εξετάσαμε παραπάνω και έτσι δεν θα εξετάσουμε τον κώδικα τους. Παρατηρούμε απλά, ότι στην τελευταία περίπτωση, όπου οι ταλαντωτές της δεύτερης σειράς του πλέγματος ταλαντωτών διαμορφώνουν αυτούς της τρίτης σειράς, η μεταβλητή που αποτελεί το τελευταίο όρισμα της μεθόδου "modulateGenerator" είναι η μεταβλητή "MODULATION2TYPE" και όχι η "MODULATION1TYPE" που ήτανε πριν.

Η τελευταία πιθανή κατάσταση του grid είναι αυτή στην οποία υπάρχουν ενεργοποιημένοι ταλαντωτές σε όλες τις γραμμές του πλέγματος ταλαντωτών. Αυτή η κατάσταση περιγράφεται από την τιμή '111' της μεταβλητής "GRIDSTATE". Ο κώδικας που αντιστοιχεί σε αυτήν την περίπτωση φαίνεται παρακάτω:

```
case 111:
        out1=AudioSynth.Osc1ON*AudioSynth.getInstance().sine1osc.generate
(AudioSynth.getInstance().osc1settings[2],AudioSynth.getInstance().osc1settings[1],AudioSynth.sinetable);
```

```
        out2=AudioSynth.Osc2ON*AudioSynth.getInstance().sine2osc.generate
(AudioSynth.getInstance().osc2settings[2],AudioSynth.getInstance().osc2settings[1],AudioSynth.sinetable);
        out3=AudioSynth.Osc3ON*AudioSynth.getInstance().sine3osc.generate
(AudioSynth.getInstance().osc3settings[2],AudioSynth.getInstance().osc3settings[1],AudioSynth.sinetable);
    out123 = out1 + out2 + out3;
        out4=AudioSynth.Osc4ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine4osc,
AudioSynth.getInstance().osc4settings[2],AudioSynth.getInstance().osc4settings[1],out123,AudioSynth.sinetable,Audio
Synth.MODULATION1TYPE);
        out5=AudioSynth.Osc5ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine5osc,
AudioSynth.getInstance().osc5settings[2],AudioSynth.getInstance().osc5settings[1],out123,AudioSynth.sinetable,Audio
Synth.MODULATION1TYPE);
        out6=AudioSynth.Osc6ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine6osc,
AudioSynth.getInstance().osc6settings[2],AudioSynth.getInstance().osc6settings[1],out123,AudioSynth.sinetable,Audio
Synth.MODULATION1TYPE);
    out456 = out4 + out5 +out6;
        out7=AudioSynth.Osc7ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine7osc,
AudioSynth.getInstance().osc7settings[0],AudioSynth.getInstance().osc7settings[1],out456,AudioSynth.sinetable,Audio
Synth.MODULATION2TYPE);
        out8=AudioSynth.Osc8ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine8osc,
AudioSynth.getInstance().osc8settings[0],AudioSynth.getInstance().osc8settings[1],out456,AudioSynth.sinetable,Audio
Synth.MODULATION2TYPE);
        out9=AudioSynth.Osc9ON*Ass1.modulateGenerator(AudioSynth.getInstance().sine9osc,
AudioSynth.getInstance().osc9settings[0],AudioSynth.getInstance().osc9settings[1],out456,AudioSynth.sinetable,Audio
Synth.MODULATION2TYPE);
    rtn= out7 + out8 + out9;
    break;
```

Η παραπάνω περίπτωση, συνδυάζει τις περιπτώσεις όπου η μεταβλητή "GRIDSTATE" παίρνει τις τιμές '110' και '11'. Στον παραπάνω κώδικα υπολογίζονται καταρχήν οι έξοδοι των ταλαντωτών της πρώτης γραμμής του πλέγματος ταλαντωτών. Το άθροισμα αυτών διαμορφώνει στη συνέχεια τους ταλαντωτές της δεύτερης γραμμής. Το είδος αυτής της διαμόρφωσης θα καθορίζεται από την μεταβλητή "MODULATION1TYPE" της κλάσης "AudioSynth". Τέλος, το άθροισμα των εξόδων των ταλαντωτών της δεύτερης γραμμής διαμορφώνει τους ταλαντωτές της τελευταίας γραμμής του grid. Ο τύπος αυτής της διαμόρφωσης καθορίζεται από την μεταβλητή "MODULATION2TYPE" της κλάσης "AudioSynth".

Οι έξοδοι των ταλαντωτών της πρώτης γραμμής του πλέγματος ταλαντωτών καθώς και το άθροισμα τους υπολογίζονται ακριβώς με τον ίδιο τρόπο όπως και στην περίπτωση '110'. Οι έξοδοι των ταλαντωτών της δεύτερης γραμμής υπολογίζονται με παρόμοιο τρόπο όπως στην παραπάνω περίπτωση. Η μόνη διαφορά είναι στις τιμές του πλάτους των ταλαντωτών αφού οι αντίστοιχες μεταβλητές έχουν αντικατασταθεί από άλλες ώστε να περιέχουν και τους δείκτες της δεύτερης διαμόρφωσης. Τέλος, ο κώδικας με τον οποίο υπολογίζονται οι έξοδοι των ταλαντωτών της τρίτης γραμμής του grid, είναι ακριβώς ίδιος με αυτόν της περίπτωσης '11'.

## x. Η κλάση AudioSetup

Η μέθοδος “deGrid” που εξετάσαμε πριν καλείται μέσα από την κλάση “AudioSetup”. Με αυτήν την κλάση επιτυγχάνεται η επικοινωνία με την κάρτα ήχου. Πριν από την εξέταση αυτής της κλάσης θα γίνει μια σύντομη αναφορά σε κάποια από τα βασικά στοιχεία του πακέτου javax.sound.sampled.

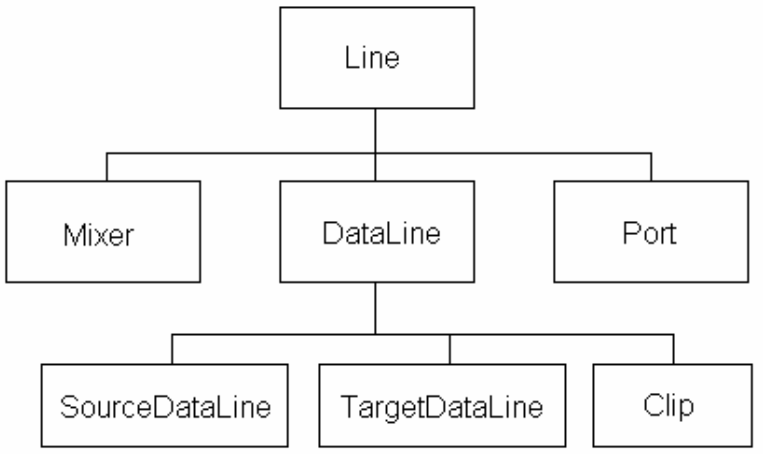
Η βασική λειτουργία του πακέτου javax.sound.sampled αφορά στη διαχείριση ροής της audio πληροφορίας. Η ροή αυτή μπορεί να γίνει είτε με τη χρήση ενός buffer (streaming), είτε αποθηκεύοντας όλα τα bytes του προς αναπαραγωγή ήχου στη μνήμη του υπολογιστή. Η εφαρμογή AudioSynth εξαιτίας του ότι είναι μια εφαρμογή πραγματικού χρόνου ,χρησιμοποιεί όπως θα δούμε τη μέθοδο “streaming”.

Η έννοια της διάταξης ήχου (audio device) αφορά συνήθως κώδικα ο οποίος επικοινωνεί με κάποια φυσική διάταξη εισόδου ή εξόδου ήχου. Οι διατάξεις ήχου στο Java Sound API, αντιπροσωπεύονται από αντικείμενα τύπου Mixer. Σκοπός ενός Mixer είναι η διαχείριση ενός ή περισσοτέρων stream εισόδου ήχου και ενός ή περισσοτέρων stream εξόδου ήχου. Με ένα τέτοιο αντικείμενο μπορούμε να «πάρουμε» τον ήχο από μια είσοδο της κάρτας ήχου μας ή να στείλουμε σήμα το οποίο μπορεί να προέρχεται από διάφορα προγράμματα, στην έξοδο της κάρτας ήχου

Στο περιβάλλον του Java Sound API, τα φυσικά audio ports μιας κάρτας ήχου όπως η είσοδος ενός μικροφώνου σε μία κάρτα ήχου, η έξοδος των ηχείων κλπ. θεωρούνται αντικείμενα τύπου Port.

Ένα αντικείμενο τύπου Mixer δέχεται ή στέλνει δεδομένα μέσω αντικειμένων που προέρχονται από το interface DataLine. Τα αντικείμενα αυτά χωρίζονται σε τρία υπο-interface: SourceDataLine, TargetDataLine και Clip. Ένα ή περισσότερα αντικείμενα τύπου SourceDataLine δέχονται δεδομένα ήχου τα οποία με τη σειρά τους παρέχουν σε ένα αντικείμενο Mixer. Αντίστοιχα ένα αντικείμενο Mixer στέλνει δεδομένα ήχου σε ένα ή περισσότερα αντικείμενα τύπου TargetDataLine. Τα αντικείμενα SourceDataLine και TargetDataline αφορούν σε stream ήχου. Αντίθετα, ένα αντικείμενο τύπου Clip αφορά πληροφορία ήχου η οποία είναι ήδη ομαδοποιημένη -όπως ένα αρχείο ήχου. Έτσι ένα Clip έχει ενσωματωμένα τα δεδομένα του ήχου και μπορούμε να τα ανακτούμε ή να τα αναπαράγουμε όσο συχνά θελήσουμε.

Το παρακάτω διάγραμμα δείχνει την ιεραρχία των επτά interface που αποτελούν το πακέτο javax.sound.sampled.




Όπως φαίνεται από το παραπάνω διάγραμμα το interface Line αποτελεί την κορυφή της ιεραρχίας του πακέτου javax.sound.sampled. Ένα αντικείμενο Line μπορεί να είναι ένας οποιοσδήποτε πόρος ο οποίος αποτελεί προορισμό ή προέλευση ψηφιακών δειγμάτων ήχου. Με αυτήν την έννοια όλα τα παραπάνω αντικείμενα αποτελούν αντικείμενα τύπου Line.

Σε αυτό το σημείο πρέπει να αναφερθούμε στην κλάση "Info" η οποία αποτελεί εσωτερική κλάση των interface Line, Mixer, Port και DataLine (Line.Info, Mixer.Info, Port.Info και DataLine.Info). Τα αντικείμενα της κλάσης αυτής παρέχουν πληροφορίες σχετικά με τα χαρακτηριστικά καθενός από τα παραπάνω interface. Το ποια ακριβώς μπορεί να είναι η χρήση αυτών των πληροφοριών θα το δούμε παρακάτω κατά την εξέταση του κώδικα της κλάσης "AudioSetup".

Τέλος, πριν περάσουμε στην εξέταση του κώδικα της κλάσης "AudioSetup" πρέπει να αναφερθούμε στην κλάση "AudioSystem" του πακέτου javax.sound.sampled. Η Κλάση AudioSystem αποτελεί το σημείο αναφοράς των πόρων ενός συστήματος ψηφιακού ήχου. Μέσω της κλάσης αυτής μπορούμε να αναζητήσουμε τους πόρους που χρησιμοποιεί ένα συγκεκριμένο σύστημα και στη συνέχεια να χρησιμοποιήσουμε κάποιους από αυτούς. Αναφορικά, η κλάση αυτή μπορεί να αναζητήσει πληροφορία για τους ακόλουθους πόρους: Mixers, Lines, μετατροπές format ήχου , αρχεία και streams.

Σε αυτό το σημείο θα ξεκινήσουμε την ερμηνεία του κώδικα της κλάσης AudioSetup. Παρακάτω φαίνονται οι πρώτες γραμμές του.

```
package gr.teicrete.audiosynth;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.SourceDataLine;

import java.io.ByteArrayInputStream;
```

```
import java.io.IOException;
```

Στην πρώτη γραμμή κώδικα βλέπουμε την δήλωση του πακέτου στο οποίο ανήκει η κλάση, ενώ στις επόμενες βλέπουμε τις κλάσεις που κάνουμε "import" προκειμένου να ορίσουμε την κλάση. Παρατηρούμε ότι οι περισσότερες από αυτές ανήκουν στο πακέτο javax.sound.sampled.

Παρακάτω βλέπουμε τη δήλωση της κλάσης και το μπλοκ αρχικοποίησης το οποίο συνοδεύεται από επεξηγηματικά σχόλια.

```
public class AudioSetup {

        AudioFormat audioFormat = new AudioFormat (AudioFormat.Encoding.PCM_SIGNED ,44100 , 16 , 1 , 2,
44100 , false); // αντικείμενο το οποίο καθορίζει το format του προς αναπαραγωγή ήχου
        AudioInputStream audiostream;// Αντικείμενο αναφοράς στο audio stream
        SourceDataLine  line = null; // αντικείμενο SourceDataLine
        DataLine.Info      info = new DataLine.Info(SourceDataLine.class,audioFormat);
//Αντικείμενο της κλάσης DataLineInfo
        private byte [] bytedata = new byte [2];//πίμνακας byte δύο θέσεων
```

Από τα αντικείμενα που αρχικοποιούνται στο παραπάνω μπλοκ αξίζει να αναφερθούμε στα αντικείμενα audioFormat και info. Για το αντικείμενο audioFormat ισχύουν ακριβώς τα ίδια όπως και για το αντίστοιχο αντικείμενο που χρησιμοποιήσαμε στην κλάση "SineGenerator". Επαναλαμβάνουμε ότι περιγράφει μονοφωνικό ήχο 16 bit, με PCM κωδικοποίηση, little endian. Το αντικείμενο info της κλάσης DataLine.Info αποτελεί ένα αντικείμενο που περιέχει πληροφορίες για μια SourceDataLine στην οποία θα «εγγράφεται» ήχος με το format που ορίζεται από το αντικείμενο "audioFormat".

Παρακάτω υπάρχει η μέθοδος – δημιουργός της κλάσης.

```
  protected AudioSetup() {
        try {
                line = (SourceDataLine) AudioSystem.getLine(info);
                line.open(audioFormat , 8192);
                }
        catch (Exception e){
                        e.printStackTrace();

                }
                line.start();
  }
```

Μέσα στο μπλοκ "try" παρατηρούμε καταρχήν την εξής γραμμή κώδικα: line = (SourceDataLine) AudioSystem.getLine(info); Με αυτόν τον τρόπο επιστρέφεται ένα αντικείμενο SourceDataLine οποίο αντιστοιχεί στην περιγραφή που γίνεται μέσω του αντικειμένου "info" και την οποία εξετάσαμε παραπάνω. Με την μέθοδο "open" που καλούμε στη συνέχεια για το παραπάνω αντικείμενο έχουμε αποκλειστική πρόσβαση και είμαστε έτοιμοι να χρησιμοποιήσουμε το παραπάνω αντικείμενο "line" .Στα ορίσματα αυτής της μεθόδου δηλώνουμε το format του ήχου που θα σταλεί στο SourceDataLine αντικείμενο και το μέγεθος του byte στο οποίο θα αποθηκεύεται το stream του ήχου. Το

μέγεθος του buffer δηλώνεται σε bytes και ο αριθμός αυτός πρέπει να συμπίπτει με κάποιο ακέραιο πολλαπλάσιο των frame του προς αναπαραγωγή ήχου. Γενικά, όσο μικρότερο είναι το μέγεθος του buffer θα έχουμε και μικρότερο latency. Το πρόβλημα με τα μικρά μεγέθη buffer είναι πως υπάρχει μεγάλη πιθανότητα τα δεδομένα να μην εγγραφούν σωστά, με αποτέλεσμα να έχουμε ασυνέχειες στον ήχο. Επιπλέον, προκειμένου ένα μικρό buffer να γεμίζει διαρκώς, καταναλώνεται περισσότερος χρόνος από τη CPU με αποτέλεσμα να πέφτει η συνολική απόδοση του συστήματος. Στην συγκεκριμένη περίπτωση έχουμε δώσει την τιμή 8192. Το μπλοκ "try" υπάρχει γιατί σε κάποιο σύστημα είναι πιθανό να μην είναι διαθέσιμη μια SourceDataLine όπως αυτή που περιγράφουμε. Το τι θα συμβεί σε τέτοιες περιπτώσεις περιγράφεται από το μπλοκ "catch" που ακολουθεί. Στην τελευταία γραμμή κώδικα της μεθόδου – δημιουργού, καλούμε την μέθοδο "start" και πάλι για το αντικείμενο "line". Με αυτόν τον τρόπο ενεργοποιούμε τη SourceDataLine ώστε να είμαστε έτοιμοι στη συνέχεια να «εγγράψουμε» εκεί το stream του ήχου.

Στις γραμμές κώδικα που ακολουθούν βλέπουμε τη μέθοδο "builtAudioBytes" της κλάσης AudioSetup.

```
protected byte[]  buildAudioBytes (GridDecoder GD1){
    int Value=(int)(GD1.deGrid()* AudioSynth.getInstance().generalsettingspanel.getGlobalAmplitude());
    AudioSynth.getInstance().generalsettingspanel.setPeak((double)Value/(double)32767);
    //this is for 16 bit mono, little endian
    bytedata[ 0] = (byte) (Value & 0xFF);
    bytedata[ 1] = (byte) ((Value >>> 8) & 0xFF);
    return bytedata;
    }
```

Η μέθοδος αυτή δέχεται σαν όρισμα ένα αντικείμενο της κλάσης GridDecoder και επιστρέφει έναν πίνακα byte. Όταν καλείται αυτή η μέθοδος μέσα από την κλάση AudioSynth σαν όρισμα έχουμε όπως είναι αναμενόμενο το μοναδικό αντικείμενο της κλάσης "GridDecoder".

Στην πρώτη γραμμή της μεθόδου βλέπουμε τον υπολογισμό της τιμής μιας μεταβλητής με το όνομα "Value". Η τιμή αυτή δίνεται από το γινόμενο της τιμής που επιστρέφεο η μέθοδος "deGrid" της κλάσης "GridDecoder" επί της τιμής που επιστρέφει η μέθοδος "getGlobalAmplitude" της κλάσης "generalsettingspanel". Έτσι η τιμή "Value" αντιπροσωπεύει σε κάθε στιγμή την έξοδο του κυκλώματος που υλοποιήσαμε με το πλέγμα ταλαντωτών πολλαπλασιασμένη με το συνολικό πλάτος του AudioSynth. Η επόμενη γραμμή κώδικα χρησιμοποιεί την παραπάνω τιμή ώστε να εμφανιστεί στο σχετικό πεδίο του πλαισίου γενικών ρυθμίσεων το peak του συνολικού πλάτους του σήματος εξόδου. Αυτό επιτυγχάνεται όπως έχουμε ήδη δει με κλήση στη μέθοδο "setPeak" της κλάσης generalsettingspanel.

Αυτό που απομένει για την συγκεκριμένη μέθοδο, είναι να χρησιμοποιήσει την τιμή "Value" ώστε να δημιουργήσει τα bytes που θα αναπαριστούν το σήμα εξόδου. Αυτό επιτυγχάνεται με τις γραμμές κώδικα που βλέπουμε παρακάτω:

```
//this is for 16 bit mono, little endian
```

```
bytedata[ 0] = (byte) (Value & 0xFF);
bytedata[ 1] = (byte) ((Value >>> 8) & 0xFF);
```

Με αυτόν τον κώδικα δημιουργούμε δύο bytes με τη χρήση της μεταβλητής "Value" και τα «τοποθετούμε» στον πίνακα δύο θέσεων "bytedata". Οι δύο θέσεις αυτού του πίνακα αποθηκεύουν ένα sample ηχητικής πληροφορίας. Αυτός είναι και ο πίνακας που επιστρέφεται από την μέθοδο.

Στις παρακάτω γραμμές κώδικα βλέπουμε και πάλι πως καλούμε τις δύο μεθόδους της κλάσης "AudioSetup" μέσα από την κλάση "AudioSynth".

```
while(playing) {
        g.writeBytestoLine(g.buildAudioBytes(GD1));
}
```

Ο επιστρεφόμενος πίνακας της μεθόδου "buildAudioBytes" που μόλις εξετάσαμε, αποτελεί το όρισμα για την κλήση στη μέθοδο "writeBytestoLine" ο κώδικας της οποίας φαίνεται παρακάτω:

```
protected void  writeBytestoLine (byte [] bytedata){
    AudioInputStream audiostream =new AudioInputStream((new
    ByteArrayInputStream ( bytedata)), audioFormat , bytedata.length );
    try {
        int numbytesread;
        while ( (numbytesread = audiostream.read(bytedata ) ) != -1) {
            line.write(bytedata,0,numbytesread);
        }
    } catch(IOException e) {
        e.printStackTrace();
    }

}
```

Στην πρώτη γραμμή του κώδικα δημιουργείται ένα αντικείμενο της κλάσης "AudioInputStream". Κάθε αντικείμενο αυτής της κλάσης αποτελεί ένα audio stream με καθορισμένο audio format και μέγεθος. Στο πρώτο όρισμα της μεθόδου – δημιουργού εισάγουμε το stream στο οποίο βασίζεται το αντικείμενο της κλάσης "AudioInputStream" που θέλουμε να δημιουργήσουμε. Στην συγκεκριμένη περίπτωση το stream είναι ένα αντικείμενο της κλάσης "ByteArrayInputStream". Τα αντικείμενα της κλάσης "ByteArrayInputStream" περιέχουν έναν εσωτερικό buffer ο οποίος περιέχει bytes τα οποία μπορούν να "διαβαστούν" από το stream. Στην συγκεκριμένη περίπτωση έχουμε σαν buffer τον πίνακα που «εξήχθη» από την μέθοδο "buildAudioBytes". Το δεύτερο όρισμα της μεθόδου – δημιουργού της κλάσης "AudioInputStream" καθορίζει το format του ήχου του stream ενώ το τρίτο καθορίζει το μέγεθος του stream. Το μέγεθος εκφράζεται σε sampleframes και όχι σε bytes.

Οι κύριες εργασίες της μεθόδου καθορίζονται μέσα στο μπλοκ "try" που ακολουθεί. Στην πρώτη γραμμή αυτού του μπλοκ κώδικα αρχικοποιείται μια μεταβλητή ακεραίων με το όνομα "numbytesread". Στη συνέχεια ακολουθεί μια ένας βρόχος "while" με τη

συνθήκη: numbytesread = audiostream.read(bytedata) . Η κλήση της μεθόδου “read” της κλάσης “AudioInputStream” «διαβάζει» έναν αριθμό bytes από το αντικείμενο της κλάσης “AudioInputStream” και τα αποθηκεύει στον πίνακα – buffer που δίνεται στο όρισμα. Όταν η επιστρεφόμενη τιμή είναι ‘-1’ σημαίνει ότι έχει διαβαστεί ολόκληρο το stream. Ο εσωτερικός κώδικας του βρόχου “while”, είναι ο εξής: line.write(bytedata,0,numbytesread); .Η μέθοδος “write” «γράφει» τα audio δεδομένα που περιέχονται στον πίνακα του πρώτου ορίσματος της, στο “mixer” μέσω του αντικειμένου “line” της κλάσης SourceDataLine. Στην μέθοδο – δημιουργό αποκτήσαμε πρόσβαση στο αντικείμενο “line” μέσω αναφοράς στην κλάση “AudioSystem” χωρίς να αναφερθούμε σε κάποιο αντικείμενο της κλάσης “Mixer”. Ωστόσο το ότι κάναμε αυτήν την παράκαμψη δεν σημαίνει ότι το αντικείμενο “line” δεν αντιστοιχεί σε κάποιο “mixer”, δηλαδή σε κάποια διάταξη ήχου. Το δεύτερο ακέραιο όρισμα της μεθόδου “write” αφορά στο offset. Σε περίπτωση που θέλαμε να μην αναπαραχθούν κάποια από τα audio bytes του stream θα δίναμε μια τιμή διαφορετική από το ‘0’ που έχουμε στην συγκεκριμένη περίπτωση. Το τελευταίο όρισμα καθορίζει τον αριθμό των bytes που θέλουμε να γραφούν στη SourceDataLine και όπως βλέπουμε θα ισούνται με την τιμή της μεταβλητής “numbytesread” που εξετάσαμε παραπάνω.

Τα μπλοκ “try” και “catch” της μεθόδου υπάρχουν και πάλι με σκοπό τη διαχείρηση σφαλμάτων σε περίπτωση που δεν μπορούμε να έχουμε πρόσβαση στο αντικείμενο “line”.

Η μέθοδος “buildAudioBytes” και η μέθοδος “writeBytestoLine”, τις οποίες εξετάσαμε παραπάνω, καλούνται περιοδικά. Ο ήχος θα αρχίσει να αναπαράγεται μόλις ο αριθμός των audio bytes που εγγράφηκαν στο “SourceDataLine” αντικείμενο γίνει ίσος με το μέγεθος του buffer που ορίσαμε για αυτό στη μέθοδο – δημιουργό της κλάσης AudioSetup.

# Βιβλιογραφία

## Java

Cadenhead, Rogers and Lemay, Laura 2003.
"Πλήρες εγχειρίδιο της Java 2"
Εκδόσεις Μ. Γκιούρδας.

Eckel, Bruce 2000.
"Thinking in Java, Second Edition" (ebook-pdf version)
Prentice Hall.

Horton, Ivor 2003.
"Beginning Java 2 SDK 1.4 Edition"
Wrox Editions.

Διαμαντόπουλος, Ταξιάρχης 2004.
"Ηλεκτρονικές Σημειώσεις στη Java"
(http://www.teiher.gr/mta/user_pages/staff/diamantopoulos/java/jnotes/j00.htm).

## JavaSound

2002.
"Java Sound Programmer Guide"
Sun Microsystems Inc.
http://java.sun.com/j2se/1.4.2/docs/guide/sound/programmer_guide/contents.html

Horton, Ivor 2003.
"Adding Sound to your Programs"
(Beginning Java 2 JDK 1.3 Edition).
Wrox Editions, pp. 921 - 967.

## Σύνθεση Ήχου (AM, FM)

Boulanger, Richard 2000.
"The Csound Book"
MIT Press,
pp. 237-242 (Ring Modulation),
249-253 (FM),
261-264 (FM),
197-206 (FM).

Dodge, Charles & Jerse, Thomas A. 1985.
"Computer Music. Synthesis, Composition, and Performance"
Schrimer Books,
pp. 80 (Modulation),
80-82 (Amplitude Modulation),

82-85 (Ring Modulation),
85 (Single Side Band Modulation),
105-128 (FM).

Moore, F.Richard 1990.
"Elements of Computer Music"
Prentice Hall,
pp. 185-187, 239 (Amplitude Modulation),
191-199, 316-332 (FM).

Roads ,Curtis 1999.
"The Computer Music Tutorial"
MIT Press,
pp 215-224 (Amplitude Modulation),
224-236 (FM),
236-239 (MC FM),
239-242 (MM FM),
242-250 (feedback FM).

Διαμαντόπουλος, Ταξιάρχης 2004.
"Προγραμματισμός & Σύνθεση Ήχου"
Εκδόσεις Έλλην,
pp. 77-107 (Σύνθεση Πίνακα Κυματομορφής),
247-271 (Διαμόρφωση Πλάτους),
273-306 (Διαμόρφωση Συχνότητας).

## Δικτυακές τοποθεσίες

http://java.sun.com

http://java.sun.com/products/java-media/sound/index.jsp

http://www.jsresources.org/