ΤΕΧΝΟΛΟΓΙΚΟ **Ε**ΚΠΑΙΔΕΥΤΙΚΟ **Ι**ΔΡΥΜΑ **Κ**ΡΗΤΗΣ
ΣΧΟΛΗ **Τ**ΕΧΝΟΛΟΓΙΚΩΝ **Ε**ΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ **Ε**ΦΑΡΜΟΣΜΕΝΗΣ **Π**ΛΗΡΟΦΟΡΙΚΗΣ & **Π**ΟΛΥΜΕΣΩΝ

Θέμα : « Εικονική Αναπαράσταση Μουσικών Κοινοτήτων Στο Διαδίκτυο »

Επιβλέπων Καθηγητής : κ. Ακουμιανάκης Δημοσθένης

Επιμέλεια : Ευθυμίου Μιχαλίτσα Α.Μ. 1220
Γρηγορακάκη Αιμιλία Α.Μ. 1292

Κρήτη, Ιανουάριος 2009

Περιεχόμενα

1. Εκφώνηση 7
2. Παραπλήσιες εργασίες 8
2.1 ABC4J 9
2.2 Impro-Visor 13
2.3 JFugue 14
2.4 JMusic 16
2.5 JFrets 17
2.6 Οδηγίες λήψης κώδικα εφαρμογών 19
2.6.1 Λήψη κώδικα JFugue Music Notepad από NetBeans 19
2.6.2 Λήψη κώδικα JFrets από NetBeans 20
3. Μουσικοί κανόνες 22
4. Προδιαγραφές πτυχιακής εργασίας 41
5. Ανάλυση του κώδικα score-editor 43
5.1 Αρχείο: Main.java 46
5.2 Αρχεία: DPianoStave,Dstave,DGrandStave,DTrebleStave 47
5.3 Αρχεία: DNoteEditor και DStaveActionHandler 51
5.4 Αρχεία: Dot1.java, Main.form, ScoreEditor.form 53
5.5 Αρχείο: ScoreEditor.java 55
6. Υλοποίηση 56
6.1 Μετρονόμος 56
6.2 Βελτίωση αρχείου: ScoreEditor.java 58
6.3 Βελτίωση αρχείου: Main.java 63
6.4 Αρχείο: Constants.java 68
6.5 Αρχείο: GuitarNeck.java 70
6.6 Αρχείο: DStave.java 79
6.7 Διάδραση ανάμεσα στην παρτιτούρα και την εικονική κιθάρα 83
6.8 Ανοιχτές χορδές 88
6.9 Δημιουργία Ant Script 96
6.10 Μετατροπή εφαρμογής σε Applet 102
6.11 Δημιουργία Ιστοσελίδας 111
7. Σύνοψη και Συμπεράσματα 120
8. Βιβλιογραφία 121
Παράρτημα 1 124

Σχήματα

Σχήμα 2-1: Παρτιτούρα με βάση το ABC4J. 9
Σχήμα 2-2: Απεικόνιση παρτιτούρας ABC4J με νότες και μουσικό κλειδί. 10
Σχήμα 2-3: Απεικόνιση παρτιτούρας ABC4J μετά την προσθήκη μπαρών. 10
Σχήμα 2-4: Απεικόνιση παρτιτούρας ABC4J μετά την προσθήκη του μέτρου. 11
Σχήμα 2-5: Απεικόνιση παρτιτούρας ABC4J μετά την εισαγωγή tie και slurs. 11
Σχήμα 2-6: Παρτιτούρα ABC4J χωρίς στοίχιση. 12
Σχήμα 2-7: Παρτιτούρα ABC4J με πλήρη στοίχιση. 12
Σχήμα 2-8: Διεπαφή της εφαρμογής Impro-Visor. 13
Σχήμα 2-9: Διεπαφή της εφαρμογής JFrets. 17
Σχήμα 2-10: Ο μετρονόμος του JFrets. 18
Σχήμα 2-11: Παράθυρο Netbeans για το checkout του JFugue Music Notepad. 20
Σχήμα 2-12: Παράθυρο Netbeans για το checkout του JFrets. 21
Σχήμα 3-1: Μουσικό πεντάγραμμο. 22
Σχήμα 3-2: Διαστήματα μουσικού πενταγράμμου. 22
Σχήμα 3-3: Κλειδιά του ΣΟΛ. 23
Σχήμα 3-4: Κλειδιά του ΦΑ. 23
Σχήμα 3-5: Κλειδιά του ΝΤΟ. 23
Σχήμα 3-6: Κλειδί του ΣΟΛ της 2ης γραμμής. 24
Σχήμα 3-7: Οι φθόγγοι του πενταγράμμου. 24
Σχήμα 3-8: Οκτάβα ή ογδόη. 24
Σχήμα 3-9: Κλίμακα (σκάλα του ΝΤΟ). 25
Σχήμα 3-10: Τόνοι και ημιτόνια. 25
Σχήμα 3-11: Σχήματα φθογγοσήμων. 26
Σχήμα 3-12: Ολόκληρο. 27
Σχήμα 3-13: Διαίρεση ολόκληρου σε 2 μισά. 27
Σχήμα 3-14: Διαίρεση ολόκληρου σε 4 τέταρτα. 28
Σχήμα 3-15: Διαίρεση ολόκληρου σε 8 όγδοα. 28
Σχήμα 3-16: Ισοδυναμίες ολόκληρου. 29
Σχήμα 3-17: Ισοδυναμίες μισού. 30
Σχήμα 3-18: Ισοδυναμίες τετάρτου. 30
Σχήμα 3-19: Ισοδυναμίες ογδόου. 31
Σχήμα 3-20: Ισοδυναμίες δεκάτου έκτου. 31
Σχήμα 3-21: Ισοδυναμία τριακοστού δευτέρου. 32
Σχήμα 3-22: Ισοδυναμία αξίας κάθε φθογγοσήμου. 32
Σχήμα 3-23: Γραφή φθογγοσήμων στο πεντάγραμμο. 33
Σχήμα 3-24: Μουσικό μέτρο στο πεντάγραμμο. 33
Σχήμα 3-25: Μουσική σύνθεση σε μέτρο 2:4. 34
Σχήμα 3-26: Ισοδυναμία ολόκληρου παρεστιγμένου. 35
Σχήμα 3-27: Ισοδυναμίες παρεστιγμένων φθογγοσήμων. 35
Σχήμα 3-28: Πίνακας σχημάτων φθογγοσήμων και παύσεων. 36
Σχήμα 3-29: Παρεστιγμένες και δις παρεστιγμένες παύσεις. 36
Σχήμα 3-30: Απεικόνιση φθογγοσήμου ΝΤΟ δίεση στο πεντάγραμμο. 37
Σχήμα 3-31: Αντιστοιχία φθογγοσήμων στην ταστιέρα της κιθάρας. 38

Σχήμα 4-1: Διεπαφή αρχικού score editor. 41
Σχήμα 5-1: Δομή αρχείων του score editor. 43
Σχήμα 5-2: Προσθήκη βιβλιοθήκης JMusic στον score editor. 44
Σχήμα 5-3: Exception του score editor. 45
Σχήμα 5-4: Uml διάγραμμα κλάσεων των staves του score editor. 47
Σχήμα 5-5: Attributes του DStave.java. 49
Σχήμα 5-6: Απεικόνιση του JComboBox του score editor. 50
Σχήμα 5-7: Παράθυρο επεξεργασίας των 51
Σχήμα 5-8: Popup menu του score editor. 52
Σχήμα 5-9: Γραφικό περιβάλλον κλάσης View. 54
Σχήμα 5-10: Παράθυρο επιλογής μουσικού οργάνου. 55
Σχήμα 5-11: Απεικόνιση κουμπιών του score editor. 55
Σχήμα 5-12: Κώδικας εισαγωγής τυχαίων νοτών στην παρτιτούρα. 55
Σχήμα 6-1: Package structure μετρονόμου. 56
Σχήμα 6-2: Παράθυρο μετρονόμου. 57
Σχήμα 6-3: Μενού με κουμπιά του score editor. 58
Σχήμα 6-4: Exception διαγραφής τελευταίας νότας σε άδεια παρτιτούρα. 58
Σχήμα 6-5: Look & Feel εφαρμογής. 63
Σχήμα 6-6: Μενού File. 64
Σχήμα 6-7: Μενού Tools. 64
Σχήμα 6-8: Μενού Help. 64
Σχήμα 6-9: Μήνυμα στο 'About' 64
Σχήμα 6-10: Παράθυρο επιλογής αρχείου για αποθήκευση. 67
Σχήμα 6-11: Οι εικόνες του Constants.java. 68
Σχήμα 6-12: Εικονική κιθάρα JFrets. 70
Σχήμα 6-13: Απεικόνιση τωνμεγεθών των κουμπιών που υλοποιήθηκαν. 70
Σχήμα 6-14: Πρώτο prototype της εικονικής κιθάρας. 70
Σχήμα 6-15: Δεύτερο prototype της εικονικής κιθάρας. 71
Σχήμα 6-16: Τρίτο prototype της εικονικής κιθάρας. 71
Σχήμα 6-17: Απεικόνιση οβάλ σχήματος στο prototype της εικονικής κιθάρας. 72
Σχήμα 6-18: Wood texture ταστίερας της κιθάρας. 72
Σχήμα 6-19: Εικονική αναπαράσταση μπράτσου κιθάρας. 72
Σχήμα 6-20: Πάνελ κουμπιών εικονικής κιθάρας. 73
Σχήμα 6-21: Πάνελ εικονικής κιθάρας. 73
Σχήμα 6-22: Νότες στην ταστιέρα της κιθάρας. 73
Σχήμα 6-23: Διεπαφή εικονικής κιθάρας. 74
Σχήμα 6-24: Μουσικά σύμβολα που έπρεπε να επαναδημιουργηθούν. 79
Σχήμα 6-25: Απεικόνιση επιλεγμένης νότας (αλλαγή χρώματος). 82
Σχήμα 6-26: Αντιστοιχία νότας Ε της 1^{η} γραμμής στην ταστιέρα της κιθάρας. 86
Σχήμα 6-27: Αντιστοιχία νότας Ε της 1^{ης} γραμμής στην εφαρμογή. 86
Σχήμα 6-28: Το RGB της πρώτης χορδής. 88
Σχήμα 6-29: Το RGB της δεύτερης χορδής. 88
Σχήμα 6-30: Το RGB της τρίτης χορδής. 89
Σχήμα 6-31: Το RGB της τέταρτης χορδής. 89
Σχήμα 6-32: Το RGB της πέμπτης χορδής. 89
Σχήμα 6-33: Το RGB της έκτης χορδής. 90

Σχήμα 6-34: Απεικόνιση 1ης χορδής ανοιχτής. 90
Σχήμα 6-35: Απεικόνιση 2ης χορδής ανοιχτής. 90
Σχήμα 6-36: Απεικόνιση 3ης χορδής ανοιχτής. 91
Σχήμα 6-37: Απεικόνιση 4ης χορδής ανοιχτής. 91
Σχήμα 6-38: Απεικόνιση 5ης χορδής ανοιχτής. 91
Σχήμα 6-39: Απεικόνιση 6ης χορδής ανοιχτής. 91
Σχήμα 6-40: Αντιστοιχία νότας E 4ου διαστήματος στην εφαρμογή. 93
Σχήμα 6-41: Αντιστοιχία νότας D στην εφαρμογή. 94
Σχήμα 6-42: Αντιστοιχία νότας C - sharp στην εφαρμογή. 95
Σχήμα 6-43: Λίστα με tasks που υποστηρίζει το script. 100
Σχήμα 6-44: Απεικόνιση εκτέλεσης εντολής 'ant clean'. 100
Σχήμα 6-45: Απεικόνιση εκτέλεσης εντολής 'ant jar'. 101
Σχήμα 6-46: Μήνυμα λάθους της Java Console. 103
Σχήμα 6-47: Exception: diamouses.ui.scoreEditor.Main cannot be cast to java.applet.Applet. 104
Σχήμα 6-48: Στοιχεία που ακολουθούν το κλειδί ασφαλείας. 105
Σχήμα 6-49: Αρχείο .keystore. 106
Σχήμα 6-50: Αρχείο scoreeditor.cer. 106
Σχήμα 6-51: Μήνυμα στο χρήστη που αφορά τα δικαιώματα του Applet. 108
Σχήμα 6-52: Παράθυρο 'More Information'. 109
Σχήμα 6-53: Στοιχεία που συνοδεύουν το πιστοποιητικό ασφαλείας. 109
Σχήμα 6-54: Απεικόνιση Save Dialog για αποθήκευση παρτιτούρας. 110
Σχήμα 6-55: Περιβάλλον διαχείρισης Joomla. 111
Σχήμα 6-56: Template ιστοσελίδας. 112
Σχήμα 6-57: Κεντρικό μενού της σελίδας. 113
Σχήμα 6-58: Κεντρική σελίδα. 114
Σχήμα 6-59: Σελίδα 'Είσοδος στην τάξη'. 115
Σχήμα 6-60: Πίνακας ελέγχου διαχειριστή φόρουμ. 116
Σχήμα 6-61: Σελίδα 'Φόρουμ'. 117
Σχήμα 6-62: Φόρμα εγγραφής 117
Σχήμα 6-63: Σελίδα 'Βοήθεια'. 119

Πίνακες

Πίνακας 2-1: Παραπλήσια πακέτα λογισμικού στη μουσική τεχνολογία. 8
Πίνακας 2-2: Λίστα μουσικών οργάνων JFugue. 15
Πίνακας 3-1: Διεθνής ονοματολογία ακκόρντων. 39
Πίνακας 3-2: Είδη ακκόρντων. 39
Πίνακας 3-3: Σημειογραφίες ακκόρντων. 40
Πίνακας 6-1: Λίστα των tasks που υλοποιούνται στο αρχείο build.xml. 97
Πίνακας 6-2: Δικαιώματα πρόσβασης στις Δημόσιες Συζητήσεις. 118
Πίνακας 6-3: Επίπεδο πρόσβασης χρηστών στις σελίδες. 119

1. Εκφώνηση

Η πτυχιακή αυτή εργασία έχει σκοπό τη δημιουργία μιας εφαρμογής σε Java όπου ο χρήστης θα μπορεί να συνθέτει μουσική και να την αναπαράγει. Η εφαρμογή χωρίζεται σε δυο μέρη.

Το πρώτο κομμάτι αφορά την σύνθεση μουσικής σε παρτιτούρα, ώστε ο χρήστης να εξοικιωθεί με τους βασικούς κανόνες της μουσικής. Η εφαρμογή θα δίνει τη δυνατότητα στο χρήστη να εισάγει νότες,παύσεις, καθώς και να καθορίζει το μουσικό όργανο με το οποίο επιθυμεί να συνθέσει. Επίσης θα μπορεί να αναπαράγει τη μουσική που σύνθεσε.

Το δεύτερο κομμάτι αφόρα την οπτική αναπαράσταση του μουσικού οργάνου και την οπτική απεικόνιση των νοτών σε αυτό. Ο χρήστης θα έχει την δυνατότητα να επιλέγει μια νότα από την παρτιτούρα και να βλέπει τη θέση που έχει η συγκεκριμένη νότα στο μουσικό όργανο. Αυτό έχει ως σκοπό την εύκολη κατανόηση και εκμάθηση της θεωρίας της μουσικής και την αντιστοιχία με το μουσικό όργανο.

Η πτυχιακή αυτή εργασία έχει υλοποιηθεί για μουσικά όργανα που ανήκουν στην κατηγορία "κιθάρα".

Η διαδικασία ολοκλήρωσης της εργασίας αποτελείται από τα ακόλουθα βήματα:

α) Αναζήτηση εργαλείων ανοιχτού κώδικα και μελέτη των δυνατοτήτων τους,
β) Μελέτη της υπάρχουσας εφαρμογής (score editor),
γ) Σχεδιασμός της νέας εφαρμογής
δ) Ανάπτυξη και υλοποίηση της εφαρμογής της πτυχιακής εργασίας.

2. Παραπλήσιες εργασίες

Μετά από σχετική έρευνα και μελέτη αρκετών project στον παγκόσμιο ιστό, τα περισσότερα εκ των οποίων ήταν τύπου ανοιχτού λογισμικού, θα παρουσιαστούν αυτά τα οποία είναι πιο κοντά με την πτυχιακή εργασία και μπορούν να μας βοηθήσουν για να παραχθούν ιδέες οι οποίες θα αντιγραφούν και θα υλοποιηθούν στα πλαίσια της εργασίας αυτής.

Σε γενικότερες γραμμές τα projects αυτά μπορούν να τοποθετηθούν σε δύο (2) κύριες κατηγορίες. Η πρώτη κατηγορία αφορά βιβλιοθήκες σε Java που παρέχουν μηχανισμούς για αναπαραγωγή ή σύνθεση μουσικής. Η κατηγορία αυτή θα ονομάζεται για τη συνέχεια της πτυχιακής '*μουσικές βιβλιοθήκες*'. Η δεύτερη κατηγορία αφορά ολοκληρωμένες εφαρμογές που έχουν γραφτεί από μουσικούς ή για μουσικούς για να βοηθήσουν τους χρήστες τους στην εξάσκηση των μουσικών τους ικανοτήτων.

Από την πρώτη κατηγορία θα επιλεχθεί η κατάλληλη μουσική βιβλιοθήκη ώστε να χρησιμοποιηθεί κατά την υλοποίηση της πτυχιακής αυτής εργασίας. Από τη δεύτερη κατηγορία θα παρουσιαστούν οι πιο ευφυής ιδέες και οι πιο λειτουργικές υλοποιήσεις διεπαφών των εφαρμογών με τον χρήστη.

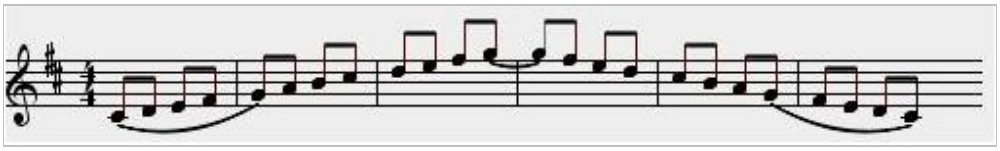
Τα πακέτα λογισμικού που πληρούν τις παραπάνω προϋποθέσεις, και επιλέχθηκαν μέσα από λίστες βιβλιοθηκών και πακέτων λογισμικού [13] παρουσιάζονται στον Πίνακα 2-1.

Όνομα	Δυνατότητες	Licence
ABC4J [2]	Κάνει parse παίζει και προβάλει παρτιτούρες κάνοντας χρήση της ABC notation [3].	GNU GPL v2
Impro-Visor [6]	Εφαρμογή για συνθέτες μουσικής jazz.	GPL
JFugue [7]	Βιβλιοθήκη για αναπαράσταση μουσικής με χαρακτήρες, και αναπαραγωγής ήχων.	Open Source
Java MIDI Kit [8]	Βιβλιοθήκη για εφαρμογές MIDI	GNU GPL
jMusic [9]	Βιβλιοθήκη	GNU GPL
jFrets [10]	Εφαρμογή για εκμάθηση κιθάρας	GNU GPL v2

Πίνακας 2-1: Παραπλήσια πακέτα λογισμικού στη μουσική τεχνολογία.

2.1 ABC4J

Από το ABC4J μπορούμε να λάβουμε ενδιαφέροντα διδάγματα μιας και το ABC notation έχει απλοποιήσει τα πράγματα πάρα πολύ. Στο Σχήμα 2-1 παρουσιάζεται παρτιτούρα με βάση το ABC4J.



Σχήμα 2-1: Παρτιτούρα με βάση το ABC4J.

Ας δούμε τα βήματα που απαιτούνται για την παραγωγή της παραπάνω παρτιτούρας. [4]

Το πρώτο βήμα είναι η δημιουργία ενός αρχείου που θα περιέχει το κλειδί και τις νότες.

```
X:0
T:A simple scale exercise
K:D
CDEFGABcdefggfedcBAGFEDC
```

Με τη χρήση του παρακάτω κώδικα μπορούμε να εμφανίσουμε την πρώτη μας παρτιτούρα.

```
import javax.swing.JFrame;

import abc.notation.Tune;
import abc.parser.TuneParser;
import abc.ui.swing.JScoreComponent;

public static void main (String[] arg) {
	String tuneAsString = "X:0\nT:A simple scale
exercise\nK:D\nCDEFGABcdefggfedcBAGFEDC\n";
	Tune tune = new TuneParser().parse(tuneAsString);
	JScoreComponent scoreUI =new JScoreComponent();
	scoreUI.setTune(tune);
	JFrame j = new JFrame();
	j.add(scoreUI);
	j.pack();
	j.setVisible(true);
}
```

Στο Σχήμα 2-2 φαίνεται η παρτιτούρα που θα προκύψει αν τρέξουμε τον παραπάνω κώδικα.

Σχήμα 2-2: Απεικόνιση παρτιτούρας ABC4J με νότες και μουσικό κλειδί.

Είναι εύκολο να παρατηρηθεί ότι το abc notation δεν ξεχωρίζει τις νότες από μόνο του. Για να βελτιώσουμε την παραπάνω παρτιτούρα πρέπει για αρχή να τοποθετηθούν μπάρες → |

```
X:0
T:A simple scale exercise
K:D
CDEF|GABc|defg|gfed|cBAG|FEDC
```

Στο Σχήμα 2-3 φαίνεται η παρτιτούρα μετά την χρήση των μπαρών.

Σχήμα 2-3: Απεικόνιση παρτιτούρας ABC4J μετά την προσθήκη μπαρών.

Το επόμενο βήμα είναι να χρησιμοποιήσουμε το time signature M:4:4 και να ομαδοποιήσουμε τις νότες ανά δύο. Το τελευταίο το επιτυγχάνουμε αφήνοντας κενά ανάμεσα στις νότες. Στο Σχήμα 2-4 απεικονίζεται η παρτιτούρα μετά την προσθήκη του μέτρου και την ομαδοποίηση.

```
X:0
T:A simple scale exercise
M:4/4
K:D
CD EF|GA Bc|de fg|gf ed|cB AG|FE DC
```

Σχήμα 2-4: Απεικόνιση παρτιτούρας ABC4J μετά την προσθήκη του μέτρου.

Το μόνο που μένει για να επιτευχθεί το επιθυμητό αποτέλεσμα είναι η εισαγωγή tie και slurs. Αυτό επιτυγχάνετε με χρήση κατάλληλων παρενθέσεων που εξηγούν από ποια νότα να ανοίγει ένα slur και σε ποια να κλείνει. Να σημειωθεί ότι για γειτονικές νότες η χρήση μιας απλής παύλας αρκεί. Έτσι χρησιμοποιώντας το παρακάτω abc notation πετυχαίνουμε το οπτικό αποτέλεσμα που φαίνεται στο Σχήμα 2-5 και επιθυμούσαμε εξ αρχής.

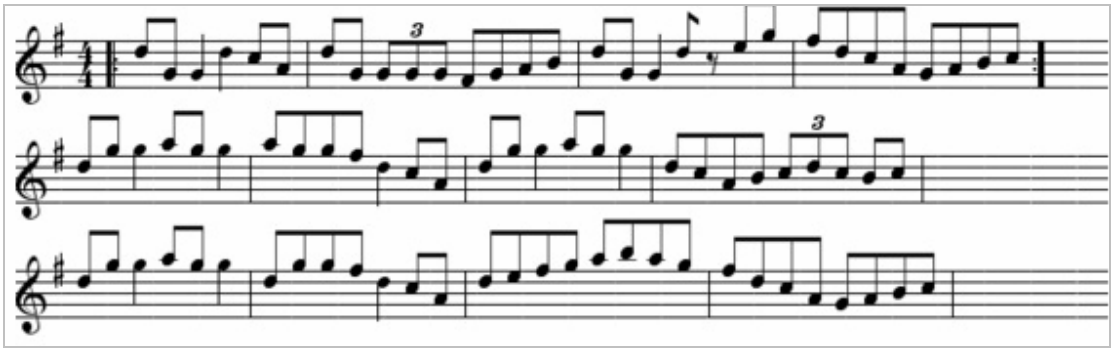
```
X:0
T:A simple scale exercise
M:4/4
K:D
(CD EF|G)A Bc|de fg-|gf ed|cB A(G|FE DC)
```

Σχήμα 2-5: Απεικόνιση παρτιτούρας ABC4J μετά την εισαγωγή tie και slurs.

Η βιβλιοθήκη abc4j παρέχει πολλές περισσότερες δυνατότητες από αυτές που ειπώθηκαν μέχρι τώρα. Πρώτον μπορεί να δημιουργήσει παρτιτούρες με χρήση Java αντικειμένων π.χ.:

```
Note note_d = new Note(Note.D);
```

Η παραπάνω δυνατότητα δεν θεωρείται κάτι το αξιοσημείωτο και δε θα γίνει επέκταση της τεχνοτροπίας στην πτυχιακή εργασία. Μια πολύ ενδιαφέρον δυνατότητα της abc4j είναι η διαχείριση παρτιτούρων.

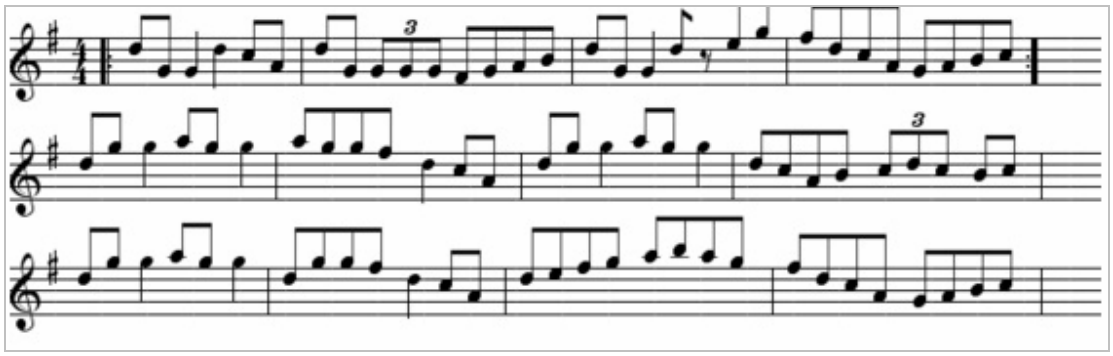


Σχήμα 2-6: Παρτιτούρα ABC4J χωρίς στοίχιση.

Η παρτιτούρα που παρουσιάζεται στο Σχήμα 2-6 θα μπορούσε να θεωρηθεί άψογη (οπτικά) από πολλές απόψεις. Όμως υπάρχει χώρος για βελτίωση της παρουσίασής της. Και συγκεκριμένα όσον αφορά στο justification. Δηλαδή στο να ξεκινάει η κάθε γραμμή της παρτιτούρας και να τελειώνει στο ίδιο σημείο. Αυτό που στα ελληνικά ονομάζουμε 'Πλήρη στοίχιση'. Αυτό μπορεί να επιτευχθεί με τη βιβλιοθήκη abc4j με τον παρακάτω κώδικα:

```
JScoreComponent jscore = new JScoreComponent();
jscore.setJustification(true);
jscore.setTune(tune);
```

Μετά την προσθήκη του παραπάνω κώδικα έχουμε το οπτικό αποτέλεσμα στο Σχήμα 2-7 που είναι πραγματικά άψογο.

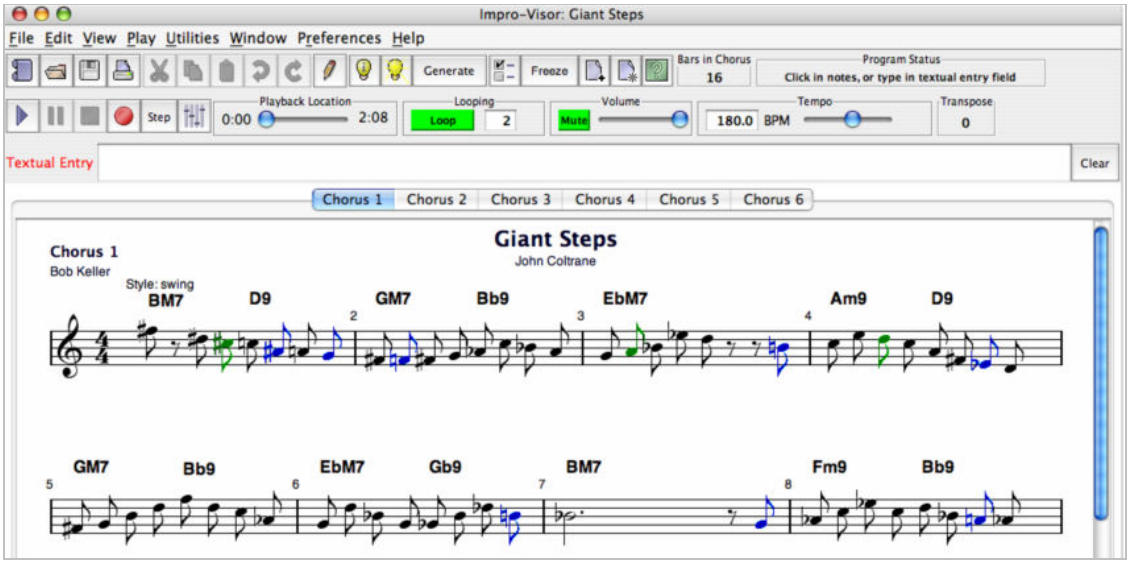


Σχήμα 2-7: Παρτιτούρα ABC4J με πλήρη στοίχιση.

Τέλος αξίζει να αναφερθεί ότι η abc4j διαθέτει δυνατότητες για αναπαραγωγή των παρτιτούρων αλλά και για μετατροπή τους σε αρχεία MIDI [5].

2.2 Impro-Visor

Η εφαρμογή Impro-Visor (βλέπε Σχήμα 2-8) είναι μια αξιέπαινη προσπάθεια που συντελείται εδώ και μερικά χρόνια στο κολλέγιο Harvey Mudd από καθηγητή και ομάδα φοιτητών που έχει καταπλήξει τον κόσμο της μουσικής τεχνολογίας και της Java. Η ομάδα αυτή έχει δημιουργήσει λογισμικό ανοιχτού κώδικα σχεδιασμένο για να βοηθάει μουσικούς jazz να συνθέτουν και να ακούν solo. Στηρίζεται στη θεωρία ότι τα ρυθμικά κομμάτια (π.χ. πιάνο, μπάσο, drums) που συνοδεύουν τη μουσική jazz μπορούν να δημιουργηθούν αυτόματα από chords.



Σχήμα 2-8: Διεπαφή της εφαρμογής Impro-Visor.

Το πλήθος των δυνατοτήτων του προγράμματος αυτό είναι τεράστιο και πιθανώς καλύπτει τις ανάγκες και του πιο απαιτητικού συνθέτη. Διαθέτει δυνατότητες για αρμονίες για διαφοροποιημένα pitches, για loops, drag and drop functionality για μεταφορά στοιχείων, αλλαγή κλειδιού και πολλά άλλα.

Ένα από τα πιο ενδιαφέροντα tricks του λογισμικού αυτού είναι η χρήση slots. Με τρόπο παρόμοιο με το Macromedia Flash, η παρτιτούρα αποτελείται από έναν αριθμό από άδεια slots μέσα στα οποία ο χρήστης μπορεί να τοποθετήσει μουσικά σύμβολα (νότες, παύσεις κτλ).

Τέλος τονίζεται ότι χρησιμοποιεί τη βιβλιοθήκη JMusic που θα αναφερθεί στη συνέχεια.

2.3 JFugue

Το JFugue είναι ένα API (Application Programming Interface) που μπορεί να χρησιμοποιηθεί για πολλαπλές χρήσεις: εμφάνιση παρτιτούρας, αναπαραγωγή και ηχογράφηση σε MIDI, χρήση πολλαπλών οργάνων και χρησιμοποιεί δική του υλοποίηση από MusicStrings.

Γενικά, αποτελεί μια ολοκληρωμένη πλατφόρμα που μπορεί να δώσει λύσεις ακόμα και στις πιο απαιτητικές εργασίες. Το βάθος στο οποίο έχει μελετηθεί η μουσική τεχνολογία και λογική είναι αξιοσημείωτο, καθώς παρέχει δυνατότητες για ταυτόχρονη χρήση πολλαπλών οργάνων.

Χρησιμοποιεί δική της υλοποίηση από MusicStrings, ένα notation παρόμοιο με το ABC notation. Το ξεπερνάει όμως από πολλές απόψεις. Άλλωστε το abc notation δημιουργήθηκε το 1991 με σκοπό τη διάδοση μελωδιών και ρυθμών που «έγραψαν ιστορία» με όσο το δυνατόν πιο απλό τρόπο. Ναι μεν στηρίζεται το abc σε κανονικοποιημένες οντότητες που γίνονται parsed και validate από BNF συντακτικούς κανόνες [11], αλλά δεν υποστηρίζει πολλαπλά voices, καθώς και ορισμό μουσικών οργάνων. Βέβαια έχουν δημιουργηθεί extensions του ABC notation που υλοποιούν τις παραπάνω δυνατότητες.

Συμπερασματικά το notation του JFugue είναι σαφώς ανώτερο και καλύπτει τεράστια γκάμα δυνατοτήτων. Στον Πίνακα 2-2 παρουσιάζεται η χαρακτηριστική λίστα από μουσικά όργανα που υποστηρίζει.

Piano	Chromatic Percussion	Organ
0 PIANO *or* ACOUSTIC_GRAND 1 BRIGHT_ACOUSTIC 2 ELECTRIC_GRAND 3 HONKEY_TONK 4 ELECTRIC_PIANO *or* ELECTRIC_PIANO1 5 ELECTRIC_PIANO2 6 HARPISCHORD 7 CLAVINET	8 CELESTA 9 GLOCKENSPIEL 10 MUSIC_BOX 11 VIBRAPHONE 12 MARIMBA 13 XYLOPHONE 14 TUBULAR_BELLS 15 DULCIMER	16 DRAWBAR_ORGAN 17 PERCUSSIVE_ORGAN 18 ROCK_ORGAN 19 CHURCH_ORGAN 20 REED_ORGAN 21 ACCORIDAN 22 HARMONICA 23 TANGO_ACCORDIAN
Guitar 24 GUITAR *or* NYLON_STRING_GUITAR 25 STEEL_STRING_GUITAR 26 ELECTRIC_JAZZ_GUITAR 27 ELECTRIC_CLEAN_GUITAR 28 ELECTRIC_MUTED_GUITAR 29 OVERDRIVEN_GUITAR 30 DISTORTION_GUITAR 31 GUITAR_HARMONICS	Bass 32 ACOUSTIC_BASS 33 ELECTRIC_BASS_FINGER 34 ELECTRIC_BASS_PICK 35 FRETLESS_BASS 36 SLAP_BASS_1 37 SLAP_BASS_2 38 SYNTH_BASS_1 39 SYNTH_BASS_2	Strings 40 VIOLIN 41 VIOLA 42 CELLO 43 CONTRABASS 44 TREMOLO_STRINGS 45 PIZZICATO_STRINGS 46 ORCHESTRAL_STRINGS 47 TIMPANI

Ensemble	Brass	Reed
48 STRING_ENSEMBLE_1 49 STRING_ENSEMBLE_2 50 SYNTH_STRINGS_1 51 SYNTH_STRINGS_2 52 CHOIR_AAHS 53 VOICE_OOHS 54 SYNTH_VOICE 55 ORCHESTRA_HIT	56 TRUMPET 57 TROMBONE 58 TUBA 59 MUTED_TRUMPET 60 FRENCH_HORN 61 BRASS_SECTION 62 SYNTHBRASS_1 63 SYNTHBRASS_2	64 SOPRANO_SAX 65 ALTO_SAX 66 TENOR_SAX 67 BARITONE_SAX 68 OBOE 69 ENGLISH_HORN 70 BASSOON 71 CLARINET
Pipe 72 PICCOLO 73 FLUTE 74 RECORDER 75 PAN_FLUTE 76 BLOWN_BOTTLE 77 SKAKUHACHI 78 WHISTLE 79 OCARINA	Synth Lead 80 LEAD_SQUARE *or* SQUARE 81 LEAD_SAWTOOTH *or* SAWTOOTH 82 LEAD_CALLIOPE *or* CALLIOPE 83 LEAD_CHIFF *or* CHIFF 84 LEAD_CHARANG *or* CHARANG 85 LEAD_VOICE *or* VOICE 86 LEAD_FIFTHS *or* FIFTHS 87 LEAD_BASSLEAD *or* BASSLEAD	Synth Pad 88 PAD_NEW_AGE *or* NEW_AGE 89 PAD_WARM *or* WARM 90 PAD_POLYSYNTH *or* POLYSYNTH 91 PAD_CHOIR *or* CHOIR 92 PAD_BOWED *or* BOWED 93 PAD_METALLIC *or* METALLIC 94 PAD_HALO *or* HALO 95 PAD_SWEEP *or* SWEEP
Synth Effects 96 FX_RAIN *OR* RAIN 97 FX_SOUNDTRACK *or* SOUNDTRACK 98 FX_CRYSTAL *or* CRYSTAL 99 FX_ATMOSPHERE *or* ATMOSPHERE 100 FX_BRIGHTNESS *or* BRIGHTNESS 101 FX_GOBLINS *or* GOBLINS 102 FX_ECHOES	Ethnic 104 SITAR 105 BANJO 106 SHAMISEN 107 KOTO 108 KALIMBA 109 BAGPIPE 110 FIDDLE 111 SHANAI	Percussive 112 TINKLE_BELL 113 AGOGO 114 STEEL_DRUMS 115 WOODBLOCK 116 TAIKO_DRUM 117 MELODIC_TOM 118 SYNTH_DRUM 119 REVERSE_CYMBAL

Πίνακας 2-2: Λίστα μουσικών οργάνων JFugue.

Η παραπάνω λίστα οργάνων έχει βρεθεί στο βιβλίο «The complete Guide to JFugue» [12]. Επίσης να τονιστεί ότι για κάθε μουσικό όργανο υπάρχει κατάλληλο notation για κάθε δυνατότητα του οργάνου. Για παράδειγμα στα drums μπορεί ο χρήστης να καθορίσει μέχρι και την ένταση με την οποία πατιέται το Foot Pedal:

X[Foot_Pedal]=1345

Εν τέλει η βιβλιοθήκη JFugue παρέχει όλες τις δυνατότητες, αλλά και πολλές παραπάνω από αυτές που παρέχει το abc4j σε όλους τους τομείς και αποτελεί σημείο εκκίνησης για οποιαδήποτε project μουσικής τεχνολογίας που χρειάζεται να υλοποιηθεί σε Java.

2.4 JMusic

Η βιβλιοθήκη της JMusic παρέχει μια σειρά από εργαλεία χρήσιμα τόσο σε συνθέτες μουσικής όσο και σε προγραμματιστές που επιθυμούν να αναπτύξουν εφαρμογές που στηρίζονται στη μουσική τεχνολογία στη Java. Επίσης παρέχει ένα ολόκληρο framework για υποβοηθούμενη σύνθεση μουσικής από υπολογιστή, για σύνθεση μουσικών οργάνων και ανάλυση μουσικής.

Παρέχει μεθόδους για οργάνωση, μετάλλαξη και ανάλυση μουσικών δεδομένων. Οι παρτιτούρες της JMusic μπορούν να γίνουν render σε MIDI αρχεία, αλλά και σε άλλα format μουσικών αρχείων για αποθήκευση και αναπαραγωγή. Μπορεί επίσης να αποθηκεύσει XML αρχεία ή ακόμα και .jm αρχεία και υποστηρίζει real-time JavaSound [14], QuickTime [15] και MidiShare [16] formats. Μάλιστα επειδή είναι γραμμένη εξ' ολοκλήρου σε Java, μπορεί να χρησιμοποιηθεί σε οποιοδήποτε λειτουργικό σύστημα.

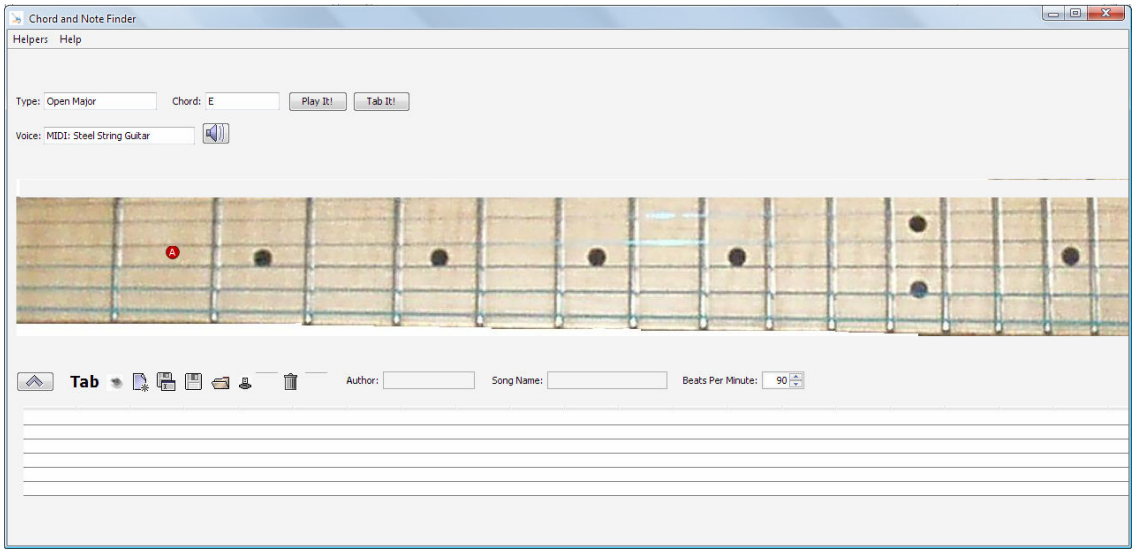
Ο αριθμός των projects που στηρίζονται στη βιβλιοθήκη αυτή είναι τεράστιος. Πέρα από το Impro-Visor που παρουσιάστηκε στο κεφάλαιο 2.2 έχει χρησιμοποιηθεί στα: CodeSounding [17], Red Wine Music [18], JM-Etude [19] και σε πολλά άλλα. Αυτό δείχνει ότι αποτελεί μια δοκιμασμένη υλοποίηση αρκετά ώριμη για να χρησιμοποιηθεί σε πραγματικές εφαρμογές.

Μετά από λεπτομερή μελέτη των διαθέσιμων τεχνολογιών, η JMusic αποτελεί μια από τις καλύτερες υποψήφιες τεχνολογίες για οποιαδήποτε εφαρμογή που χρησιμοποιεί Java Sounds, και οι δυνατότητές της θα παρουσιαστούν αναλυτικότερα στη συνέχεια της πτυχιακής.

2.5 JFrets

Το JFrets (βλέπε Σχήμα 2-9) είναι ένα πακέτο λογισμικού ανοιχτού κώδικα που δημιουργήθηκε από τον Matt Warman. Αποτελεί μια προσπάθεια για δημιουργία ενός εργαλείου που θα βοηθήσει τον χρήστη στην εκμάθηση κιθάρας.

Το πρόγραμμα αυτό μπορεί να βοηθήσει έναν αρχάριο χρήστη να εντοπίσει τις θέσεις στις οποίες πρέπει να τοποθετήσει τα δάχτυλά του για να παίξει ένα chord. Μπορεί επίσης να βοηθήσει έναν μουσικό να γράψει παρτιτούρες για τη μουσική που συνθέτει σε ηλεκτρονική μορφή.



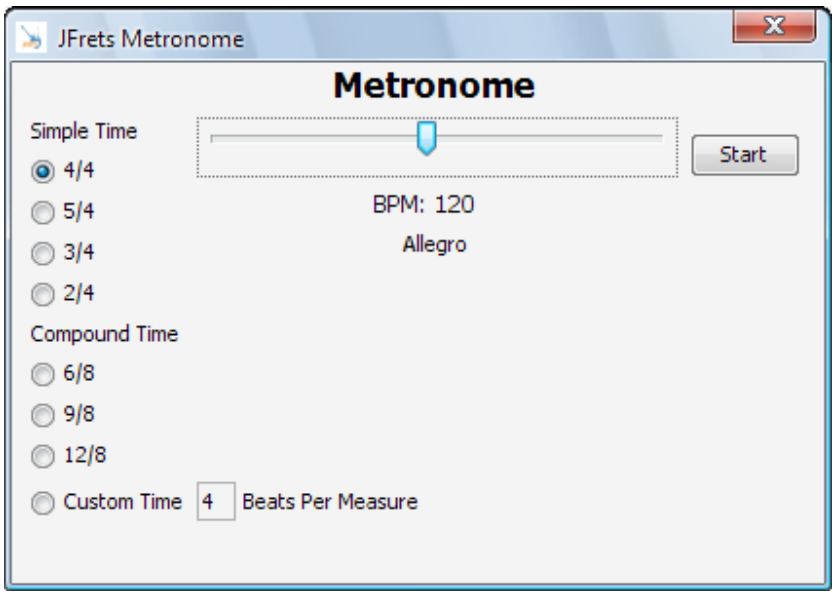
Σχήμα 2-9: Διεπαφή της εφαρμογής JFrets.

Σε αυτή την εφαρμογή ο χρήστης μπορεί να κάνει interact με το ποντίκι πάνω στην ταστιέρα της κιθάρας. Αν πατήσει αριστερό κλικ σε ένα σημείο στην ταστιέρα ακούγεται η εκάστοτε νότα στην κιθάρα και εμφανίζεται ένας κύκλος με το όνομα της νότας που ακούστηκε.

Το αξιόλογο σημείο της συγκεκριμένης εφαρμογής είναι ο τρόπος με τον οποίο υλοποιήθηκε η ταστιέρα ώστε να αλληλεπιδρά με το χρήστη. Αυτό που έγινε περιληπτικά είναι να δημιουργηθεί ένα πάνελ όπου μέσα φορτώνεται η εικόνα της ταστιέρας της κιθάρας και πάνω από αυτή υλοποιήθηκε ένα `JTable` το οποίο έχει 6 γραμμές και 23 στήλες (όσα και τα διαστήματα της ταστιέρας της κιθάρας) όπου κάθε κελί αντιστοιχεί στην νότα της κιθάρας.

Το `JTable` είναι transparent και όταν ο χρήστης πατά πάνω σε ένα σημείο στην ουσία πατά σε ένα κελί του πίνακα. Επίσης γράφεται αυτόματα και η ταμπλατούρα από κάτω.

Η εφαρμογή διαθέτει επιπλέον μετρονόμο όπως φαίνεται στο Σχήμα 2-10, για να βοηθήσει στο «συντονισμό» του μουσικού σε ορισμένο μέτρο. Μετρονόμος είναι ένα μηχάνημα που κρατάει το μέτρο στο κομμάτι που παίζει κανείς. Είναι απαραίτητο σε όλα τα μουσικά όργανα και στην ουσία παράγει ήχο κάθε συγκεκριμένα δευτερόλεπτα που καθορίζει ο χρήστης. Ο ήχος είναι σαν τους χτύπους του ρολογιού.



Σχήμα 2-10: Ο μετρονόμος του JFrets.

Επίσης διαθέτει δυνατότητες αναπαραγωγής και αποθήκευσης της μουσικής που συνθέτει ο συνθέτης.

2.6 Οδηγίες λήψης κώδικα εφαρμογών

Για να αποκτήσει κανείς τον κώδικα των παραπάνω εφαρμογών θα πρέπει να κάνει χρήση CVS [20], Subversion [21] repositories ή να τον κατεβάσει από αντίστοιχες σελίδες στο Ίντερνετ. Το CVS και το Subversion αποτελούν concurrent version systems που δίνουν τη δυνατότητα σε προγραμματιστές να εργάζονται ταυτόχρονα στον κώδικα ενός project. Για τη λήψη του κώδικα των εφαρμογών που παρουσιάστηκαν μέχρι τώρα, υπάρχει σε ορισμένα projects η δυνατότητα απόκτησης του κώδικα μόνο μέσα από τα συστήματα αυτά.

Όσον αφορά τα repositories υπάρχουν αρκετά εργαλεία τα οποία μπορούν να χρησιμοποιηθούν για τη λήψη του κώδικα. Αναγκαία προϋπόθεση είναι ο χρήστης να ανοίξει λογαριασμό στο http://dev.java.net/

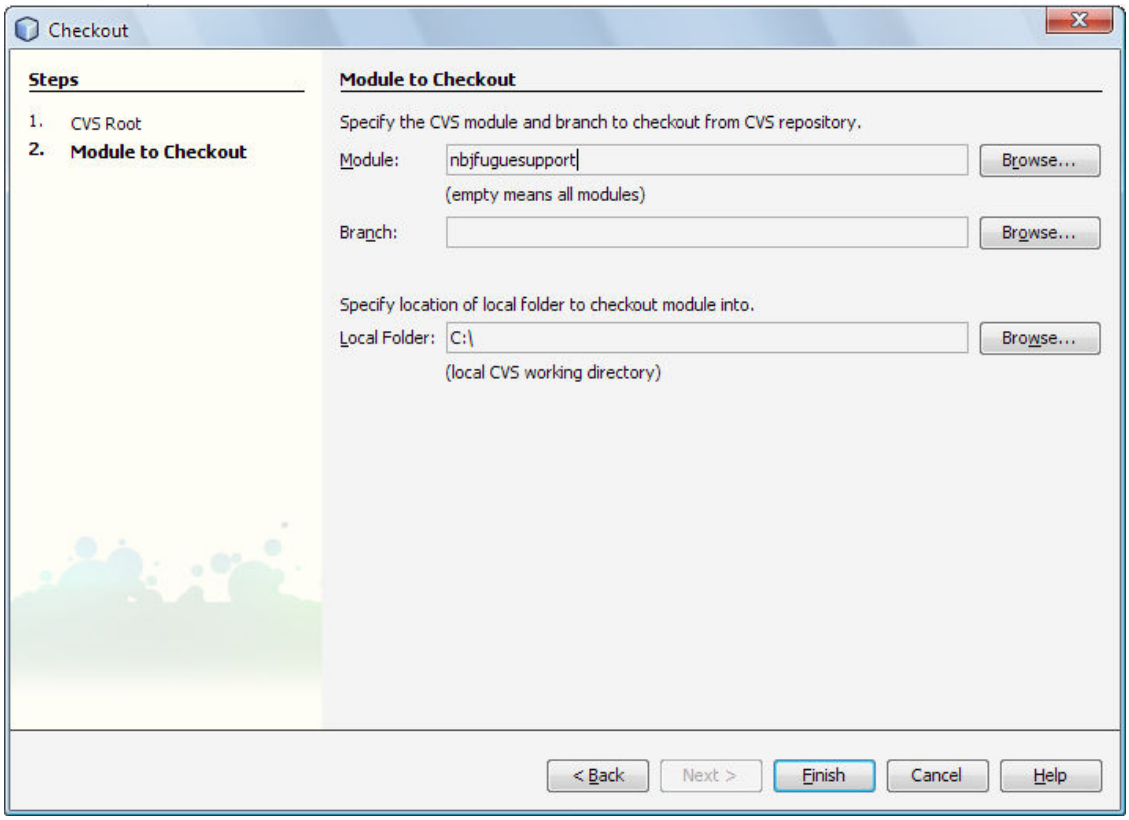
2.6.1 Λήψη κώδικα JFugue Music Notepad από NetBeans

Με τη χρήση του εργαλείου NetBeans [22] η διαδικασία είναι η ακόλουθη:

Επιλέγουμε από το μενού Versioning ⟶ CVS ⟶Checkout..

Στο παράθυρο που εμφανίζεται τοποθετούμε τα στοιχεία του λογαριασμού και του project που επιθυμούμε να κατεβάσουμε :pserver:username@cvs.dev.java.net:/cvs και πατάμε next.

Στο Σχήμα 2-11 εμφανίζεται το παράθυρο μετά την σωστή καταχώρηση των παραπάνω στοιχείων.



Σχήμα 2-11: Παράθυρο Netbeans για το checkout του JFugue Music Notepad.

Γράφετε στο πεδίο module nbjfuguesupport και στη συνέχεια επιλέγετε την τοποθεσία που θέλετε να αποθηκευτεί ο κώδικας στον υπολογιστή σας. Πατάτε finish περιμένετε μέχρι να ολοκληρωθεί το checkout και στη συνέχεια τρέχετε τον κώδικα στον υπολογιστή σας.

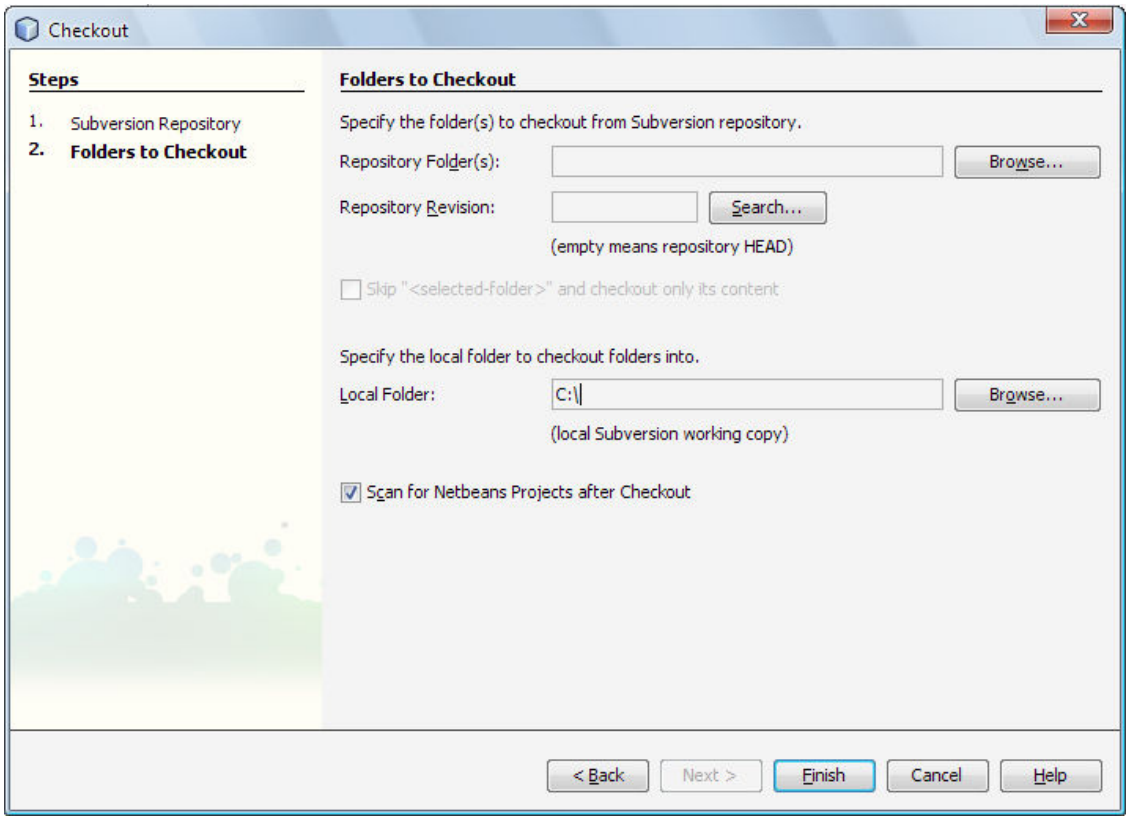
2.6.2 Λήψη κώδικα JFrets από NetBeans

Με τη χρήση του εργαλείου NetBeans η διαδικασία είναι η ακόλουθη:

Επιλέγουμε από το μενού Versioning⟶ Subversion ⟶Checkout..

Στο παράθυρο που εμφανίζεται τοποθετούμε τα στοιχεία του λογαριασμού και του project που επιθυμούμε να κατεβάσουμε https://jfrets.dev.java.net/svn/jfrets και πατάμε next.

Στο Σχήμα 2-12 παρουσιάζεται το παράθυρο μετά την σωστή καταχώρηση των παραπάνω στοιχείων.



Σχήμα 2-12: Παράθυρο Netbeans για το checkout του JFrets.

Επιλέγετε την τοποθεσία που θέλετε να αποθηκευτεί ο κώδικας στον υπολογιστή σας και πατάτε finish. Όταν ολοκληρωθεί το checkout προσθέτετε στον κώδικα της εφαρμογής την βιβλιοθήκη jdom και τρέχετε τον κώδικα.

3. Μουσικοί κανόνες

Όπως κάθε γλώσσα έχει τα γράμματα και τους κανόνες της, το ίδιο έχει και η μουσική με τη διαφορά ότι στη μουσική αντί για γράμματα χρησιμοποιούμε επτά διαφορετικούς μουσικούς ήχους (φωνές) με τις πιο κάτω ονομασίες :

Ελληνική γραφή: ΝΤΟ-ΡΕ-ΜΙ-ΦΑ-ΣΟΛ-ΛΑ-ΣΙ

Λατινική γραφή: C-D-E-F-G-A-B

Οι μουσικοί αυτοί ήχοι παράγονται από τα διάφορα μουσικά όργανα ή από τη φωνή του ανθρώπου, όταν αυτός τραγουδάει και έχουν ένα ορισμένο ύψος. Τους ήχους αυτούς τους ονομάζουμε **μουσικούς φθόγγους** και γράφονται στο πεντάγραμμο με ειδικά σχήματα, που εκφράζουν τη διάρκειά τους και λέγονται **φθογγόσημα** ή ξενόγλωσσα, **νότες** (notes).

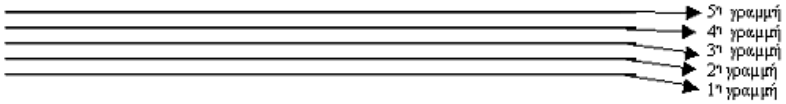
Υπάρχουν και άλλοι ήχοι που δεν έχουν ένα καθορισμένο ύψος, όπως οι θόρυβοι και οι κρότοι.

Καταλήγουμε λοιπόν στο συμπέρασμα ότι:

α) **Μουσικός φθόγγος** είναι ο ήχος (φωνή) που έχει ένα καθορισμένο ύψος.
β) **Θόρυβος - κρότος** είναι ο ήχος που δεν έχει ένα καθορισμένο ύψος.
γ) **Φθογγόσημο ή νότα** είναι το σχήμα με το οποίο γράφουμε το μουσικό φθόγγο στο πεντάγραμμο.

Πεντάγραμμο

Πεντάγραμμο είναι το σύνολο από 5 οριζόντιες και παράλληλες γραμμές που απέχουν η μια από την άλλη το ίδιο (βλέπε Σχήμα 3-1).



Σχήμα 3-1: Μουσικό πεντάγραμμο.

Από τις πέντε γραμμές του πενταγράμμου σχηματίζονται τέσσερα διαστήματα, όπως φαίνεται στο Σχήμα 3-2.

Διάστημα λέγεται η απόσταση που χωρίζει τη μια γραμμή από την άλλη.

Σχήμα 3-2: Διαστήματα μουσικού πενταγράμμου.

Από πού εξαρτάται η ονομασία των φθόγγων

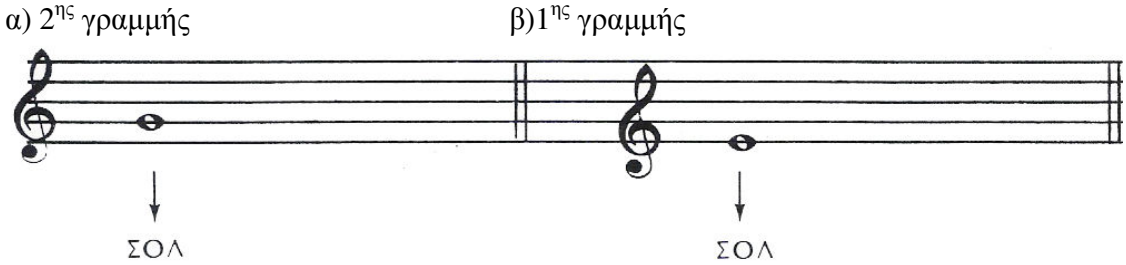
Η ονομασία των φθόγγων εξαρτάται από τη θέση που είναι γραμμένοι στο πεντάγραμμο και από το όνομα και τη θέση του Κλειδιού.

Κλειδί (Γνώμονας)

Κλειδί της μουσικής ή **γνώμονας** είναι ένα σημείο που γράφεται στην αρχή κάθε πενταγράμμου και, ανάλογα με τη θέση του, προσδιορίζει το ύψος και το όνομα ενός ορισμένου φθόγγου, και σύμφωνα με αυτόν ονομάζουμε και τους άλλους.

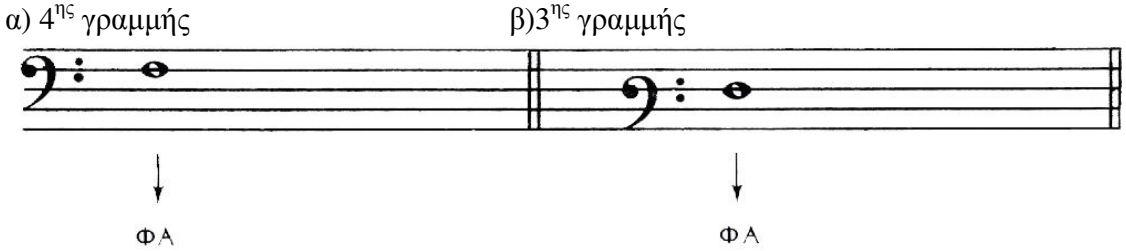
Τα Κλειδιά της μουσικής είναι τα εξής:

1. Δύο κλειδιά του ΣΟΛ (στη $2^{η}$ και στην $1^{η}$ γραμμή) τα οποία φαίνονται στο Σχήμα 3-3.



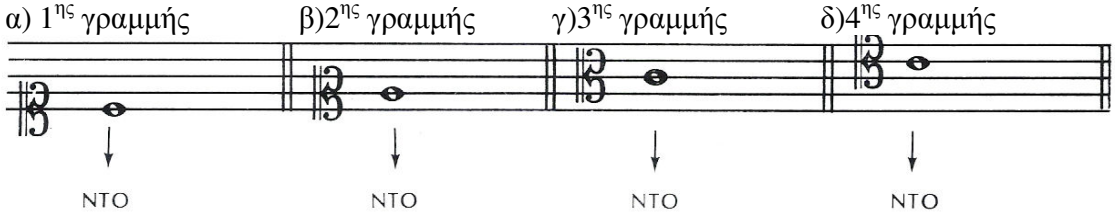
Σχήμα 3-3: Κλειδιά του ΣΟΛ.

2. Δύο κλειδιά του ΦΑ (στην $4^{η}$ και στην $3^{η}$ γραμμή) που παρουσιάζονται στο Σχήμα 3-4.



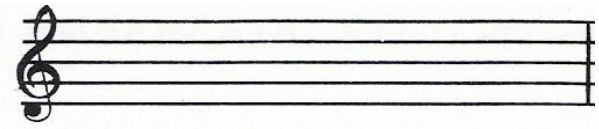
Σχήμα 3-4: Κλειδιά του ΦΑ.

3. Τέσσερα κλειδιά του ΝΤΟ(στην $1^{η}$, $2^{η}$, $3^{η}$ και $4^{η}$ γραμμή) που απεικονίζονται στο Σχήμα 3-5.



Σχήμα 3-5: Κλειδιά του ΝΤΟ.

Κλειδί του ΣΟΛ της 2ης γραμμής

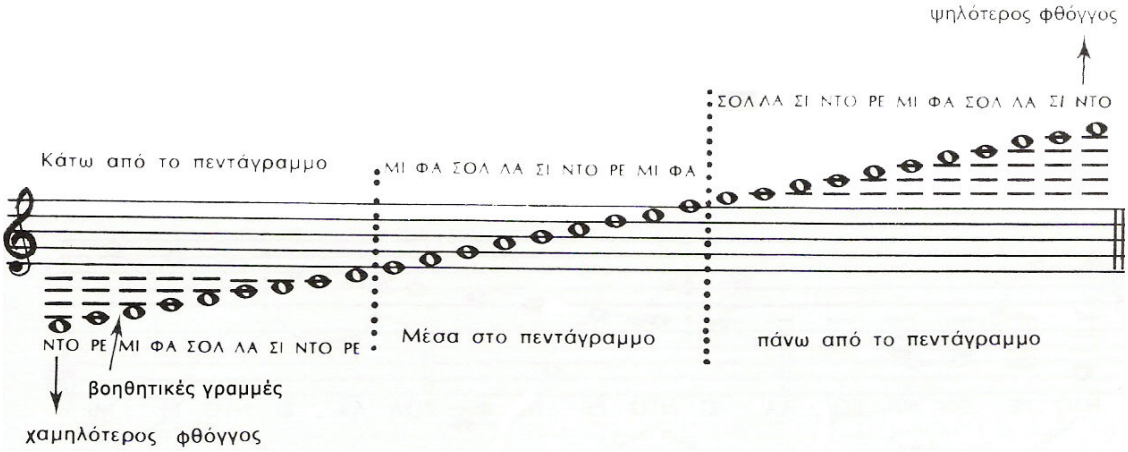


Σχήμα 3-6: Κλειδί του ΣΟΛ της 2ης γραμμής.

Λέγεται **της δεύτερης γραμμής** γιατί στο γράψιμό του κάνει ένα τύλιγμα πάνω στη δεύτερη γραμμή του πενταγράμμου, παριστάνοντας έτσι ένα φθογγόσημο στρογγυλό πάνω στη δεύτερη γραμμή (βλέπε Σχήμα 3-6).

Όλοι οι φθόγγοι

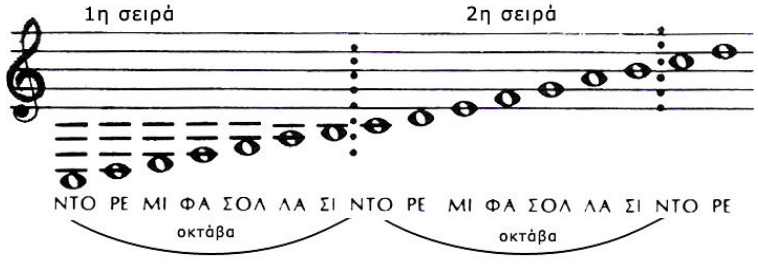
Στο Σχήμα 3-7 παρουσιάζονται όλοι οι φθόγγοι.
α) Κάτω από το πεντάγραμμο β) μέσα στο πεντάγραμμο γ) πάνω από το πεντάγραμμο



Σχήμα 3-7: Οι φθόγγοι του πενταγράμμου.

Οκτάβα (Ογδόη)

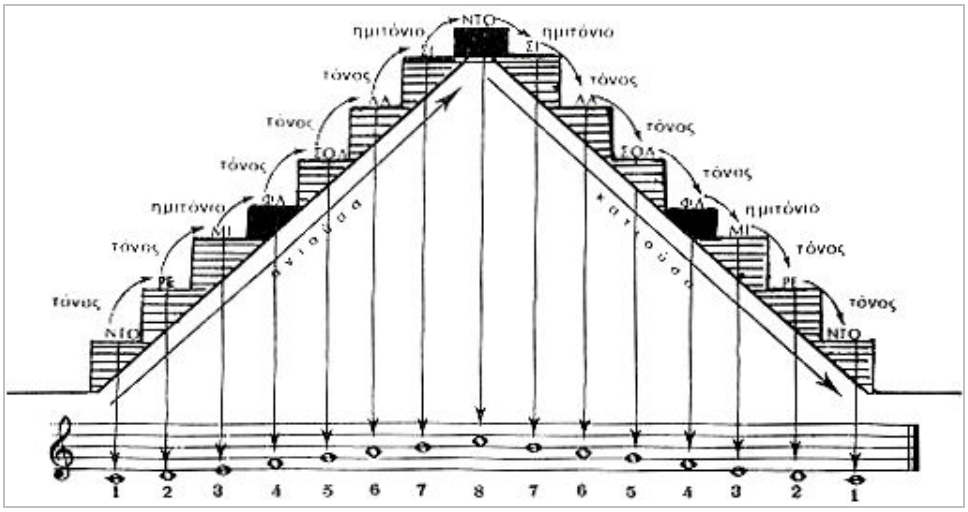
Η απόσταση ανάμεσα σε δύο φθόγγους που έχουν το ίδιο όνομα και βρίσκονται σε δύο γειτονικές σειρές λέγεται **οκτάβα (ογδόη)** γιατί η απόσταση αυτή περιλαμβάνει 8 φθόγγους, δηλαδή στο Σχήμα 3-8, η απόσταση από το ΝΤΟ της πρώτης σειράς μέχρι το ΝΤΟ της δεύτερης σειράς λέγεται οκτάβα ή ογδόη κ.ο.κ.



Σχήμα 3-8: Οκτάβα ή ογδόη.

Κλίμακα (Σκάλα)

Κλίμακα λέγεται η διαδοχή 8 συνεχών φθόγγων που ανεβαίνουν και κατεβαίνουν. Στο Σχήμα 3-9 παρουσιάζεται η κλίμακα του ΝΤΟ.

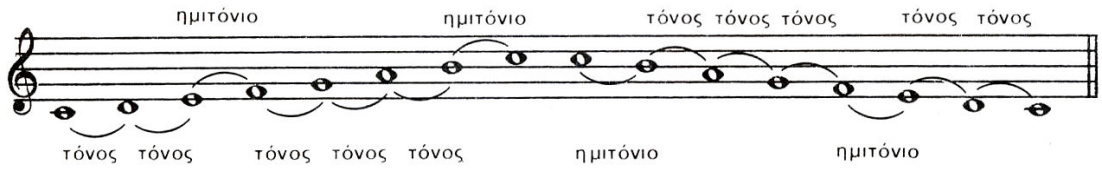


Σχήμα 3-9: Κλίμακα (σκάλα του ΝΤΟ).

Ηχητικές αποστάσεις στους φθόγγους της κλίμακας

Οι ηχητικές αποστάσεις σε δύο γειτονικούς φθόγγους της κλίμακας είναι μεγάλες και μικρές. Οι μεγάλες λέγονται τόνοι και οι μικρές ημιτόνια (βλέπε Σχήμα 3-10). **Ημιτόνιο** είναι η μισή φωνή, και **τόνος** η ολόκληρη φωνή. Ο τόνος έχει δύο ημιτόνια δηλαδή 2 μισές φωνές. Στην κλίμακα ΝΤΟ όλες οι αποστάσεις μεταξύ τους είναι μεγάλες (τόνοι), εκτός από το ΜΙ - ΦΑ και το ΣΙ - ΝΤΟ που είναι μικρές (ημιτόνια). Γι' αυτό το λόγο όταν τραγουδάμε την κλίμακα ανεβαίνοντας, στο ΜΙ - ΦΑ και στο ΣΙ - ΝΤΟ δεν υψώνουμε πολύ τη φωνή μας γιατί οι αποστάσεις αυτές είναι μικρές (ημιτόνια).

Το ίδιο γίνεται και στο κατέβασμα, που βρίσκουμε αυτούς τους φθόγγους ανάποδα, ΦΑ - ΜΙ και ΝΤΟ - ΣΙ, και εδώ δεν χαμηλώνουμε πολύ τη φωνή μας.



Σχήμα 3-10: Τόνοι και ημιτόνια.

Τις μικρές αποστάσεις τις βλέπουμε καλύτερα στο πιο πάνω παραστατικό σχήμα της κλίμακας (σκάλας), όπου την παρουσιάζουμε σαν σκάλα σπιτιού. Αν προσέξουμε, βλέπουμε ότι οι αποστάσεις από το **ΜΙ** ως το **ΦΑ** και από το **ΣΙ** ως το **ΝΤΟ** είναι μικρότερες, γιατί τα σκαλοπάτια **ΦΑ** και **ΝΤΟ** είναι κατά το μισό χαμηλότερα από τα άλλα.

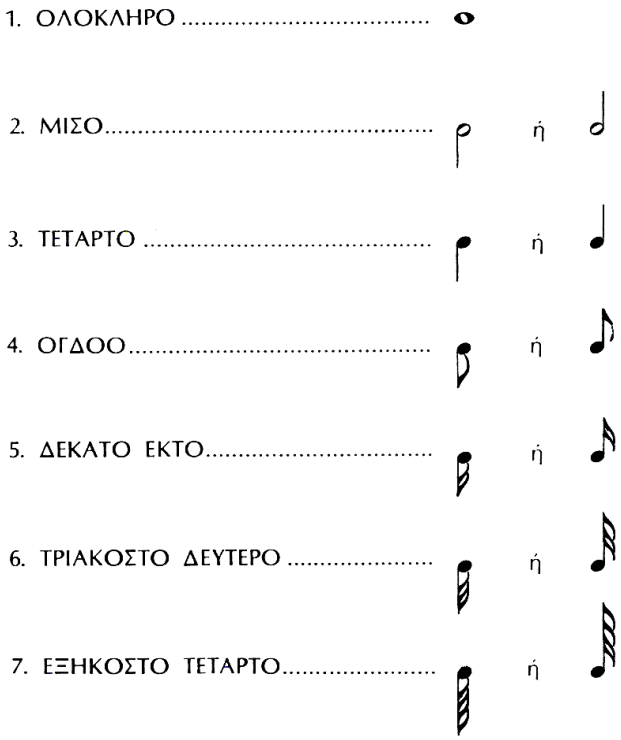
Αξίες φθογγοσήμων

Παρατηρούμε ότι οι φθόγγοι που ανήκουν σε μία μουσική σύνθεση, όταν παίζονται από διάφορα μουσικά όργανα ή τραγουδιούνται από ανθρώπινες φωνές, διαρκούν άλλοι περισσότερο και άλλοι λιγότερο. Η διάρκειά τους αυτή - η αξία τους - είναι ανάλογη με το σχήμα με το οποίο τα φθογγόσημα είναι γραμμένα στο πεντάγραμμο.

Άρα καταλήγουμε στον εξής ορισμό:

Αξίες είναι οι διάρκειες των φθόγγων που παριστάνονται στο πεντάγραμμο με διάφορα σχήματα φθογγοσήμων, και ανάλογα με το σχήμα κάθε φθογγοσήμου καθορίζεται η διάρκεια κάθε φθόγγου.

Στο Σχήμα 3-11 απεικονίζονται τα σχήματα των φθογγοσήμων. Συνολικά είναι επτά.



Σχήμα 3-11: Σχήματα φθογγοσήμων.

Διάρκεια φθογγοσήμων

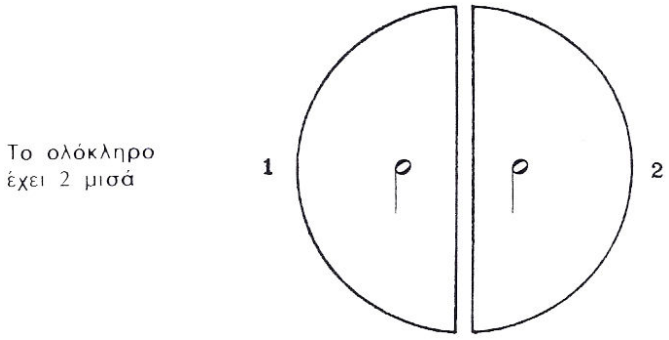
Από τα φθογγόσημα που αναφέραμε πιο πάνω, τη μεγαλύτερη διάρκεια την έχει το ολόκληρο, μετά ακολουθεί το μισό, που έχει τη μισή διάρκεια από το ολόκληρο, μετά το τέταρτο που έχει διάρκεια ίση με το ένα τέταρτο του ολόκληρου κ.ο.κ. Έτσι φτάνουμε στη μικρότερη διάρκεια, που είναι το εξηκοστό τέταρτο και έχει διάρκεια ίση με το ένα εξηκοστό τέταρτο του ολόκληρου.

Διαιρέσεις ολόκληρου και υποδιαιρέσεις αυτών

1. Ένα ολόκληρο μπορούμε να το διαιρέσουμε σε μισά, σε τέταρτα, σε όγδοα, σε δέκατα έκτα, σε τριακοστά δεύτερα και σε εξηκοστά τέταρτα. Για να καταλάβουμε πιο καλά αυτές τις διαιρέσεις, παρομοιάζουμε το ολόκληρο με ένα στρογγυλό πορτοκάλι όπως φαίνεται στο Σχήμα 3-12, π.χ.

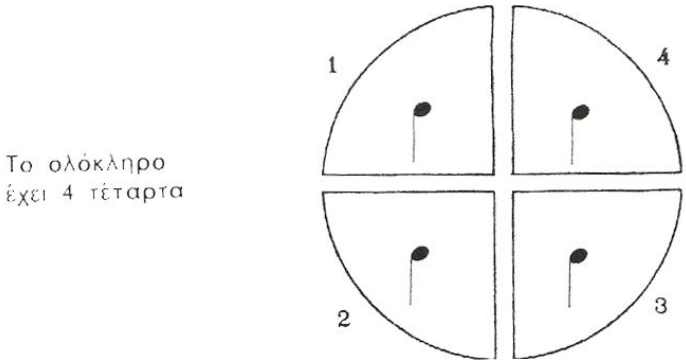
Σχήμα 3-12: Ολόκληρο.

2. Αν τώρα το πορτοκάλι το κόψουμε στη μέση όπως μας δείχνει το πιο κάτω σχήμα, θα έχουμε 2 κομμάτια από το μισό πορτοκάλι. Άρα το ολόκληρο διαιρείται σε 2 μισά όπως απεικονίζεται στο Σχήμα 3-13.



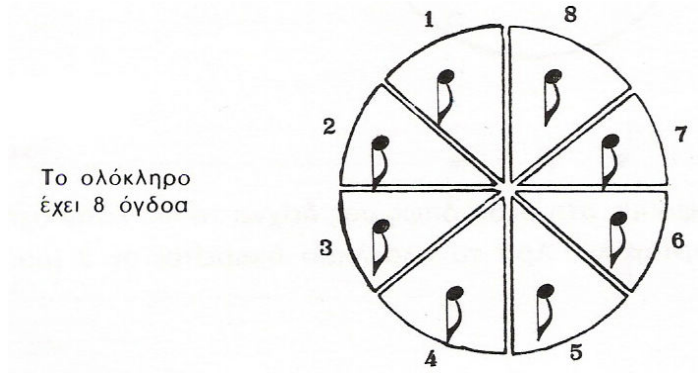
Σχήμα 3-13: Διαίρεση ολόκληρου σε 2 μισά.

3. Αν το κόψουμε στα τέσσερα θα έχουμε 4 κομμάτια και το κάθε κομμάτι το λέμε τέταρτο, όπως φαίνεται στο Σχήμα 3-14. Άρα το ολόκληρο διαιρείται σε 4 τέταρτα όπως μας δείχνει το πιο κάτω σχήμα.



Σχήμα 3-14: Διαίρεση ολόκληρου σε 4 τέταρτα.

4. Αν το κόψουμε στα οκτώ θα έχουμε 8 κομμάτια όπως απεικονίζεται στο Σχήμα 3-15, και το κάθε κομμάτι το λέμε όγδοο. Άρα το ολόκληρο διαιρείται σε 8 όγδοα όπως μας δείχνει το πιο κάτω σχήμα.



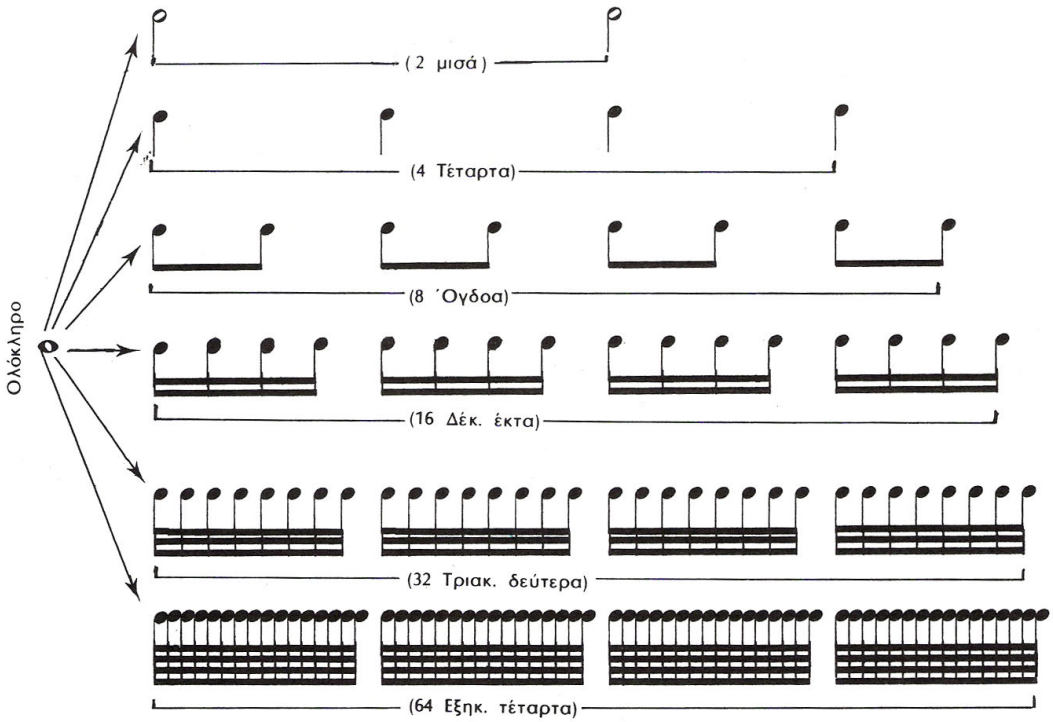
Σχήμα 3-15: Διαίρεση ολόκληρου σε 8 όγδοα.

Το ανάλογο μπορεί να γίνει αν κόψουμε το πορτοκάλι σε 16, σε 32 και σε 64 κομμάτια. Δηλαδή το ολόκληρο διαιρείται σε 16 δέκατα έκτα, ή σε 32 τριακοστά δεύτερα ή σε 64 εξηκοστά τέταρτα.

Πιο κάτω παρουσιάζουμε αναλυτικά με τι ισούται η κάθε αξία φθογγοσήμου και την κανονική τους γραφή στο πεντάγραμμο.

1. Με τι ισούται το ένα ολόκληρο

Το ολόκληρο ισούται με τα 2 μισά ($\frac{2}{2}$), ή με 4 τέταρτα ($\frac{4}{4}$), ή με 8 όγδοα ($\frac{8}{8}$), ή με 16 δέκατα έκτα ($\frac{16}{16}$), ή με 32 τριακοστά δεύτερα ($\frac{32}{32}$), ή με 64 εξηκοστά τέταρτα ($\frac{64}{64}$) (βλέπε Σχήμα 3-16).



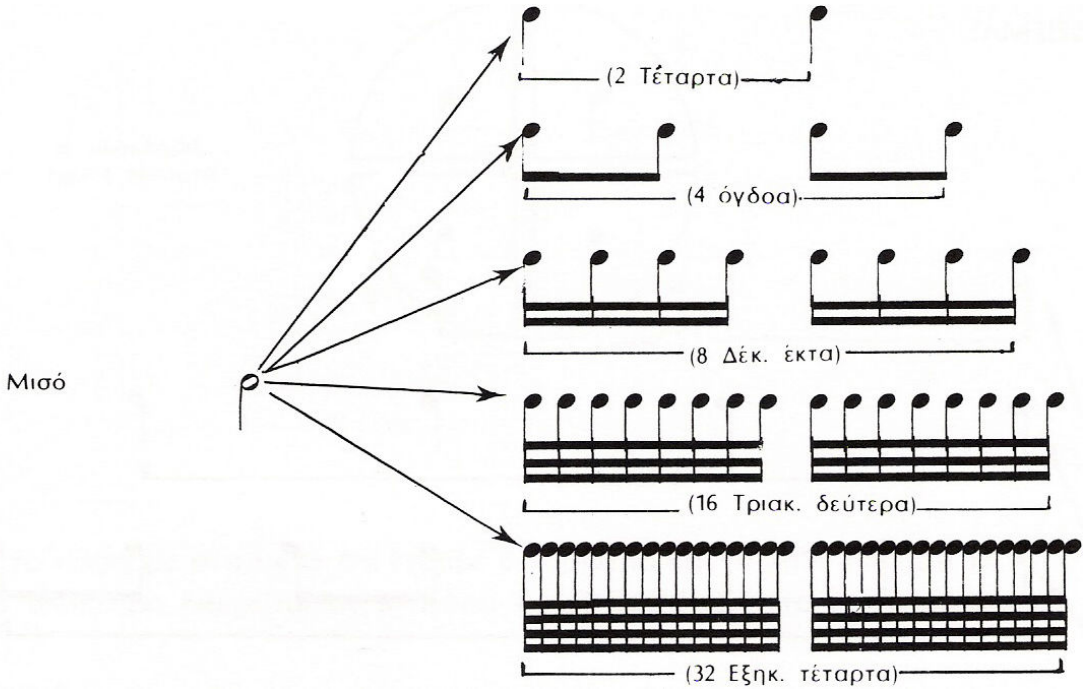
Σχήμα 3-16: Ισοδυναμίες ολόκληρου.

ΣΗΜΕΙΩΣΗ: Τα όγδοα, τα δέκατα έκτα, τα τριακοστά δεύτερα και τα εξηκοστά τέταρτα, για καλή οπτική ανάγνωση και για συντομία στο γράψιμό τους, ενώνονται μεταξύ τους με γραμμές. Με μία γραμμή τα όγδοα, με 2 τα δέκατα έκτα, με 3 τα τριακοστά δεύτερα και με 4 τα εξηκοστά τέταρτα, όπως τα βλέπουμε στο πιο πάνω σχήμα.

Έχουμε περιθώριο να ενώσουμε από δυο και πάνω.

2. Με τι ισούται το ένα μισό

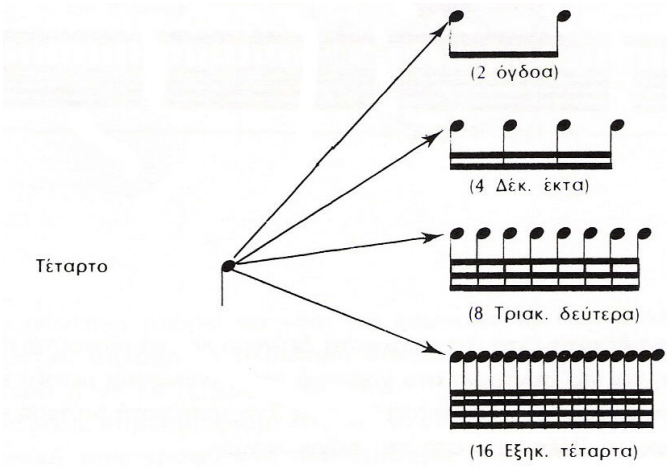
Το μισό ισούται με 2 τέταρτα ($\frac{2}{4}$), ή με 4 όγδοα ($\frac{4}{8}$), ή με 8 δέκατα έκτα ($\frac{8}{16}$), ή με 16 τριακοστά δεύτερα ($\frac{16}{32}$), ή με 32 εξηκοστά τέταρτα ($\frac{32}{64}$) (βλέπε Σχήμα 3-17).



Σχήμα 3-17: Ισοδυναμίες μισού.

3. Με τι ισούται το ένα τέταρτο

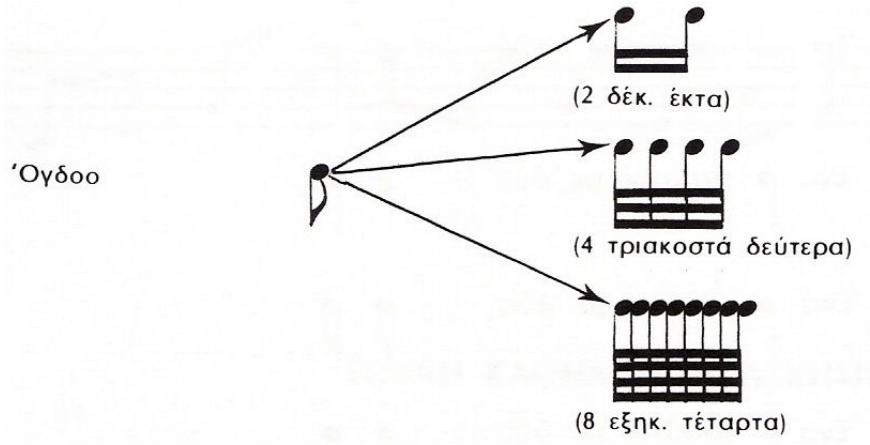
Το ένα τέταρτο ισούται με 2 όγδοα ($\frac{2}{8}$), ή με 4 δέκατα έκτα ($\frac{4}{16}$), ή με 8 τριακοστά δεύτερα ($\frac{8}{32}$), ή με 16 εξηκοστά τέταρτα ($\frac{16}{64}$)(βλέπε Σχήμα 3-18).



Σχήμα 3-18: Ισοδυναμίες τετάρτου.

4. Με τι ισούται το ένα όγδοο

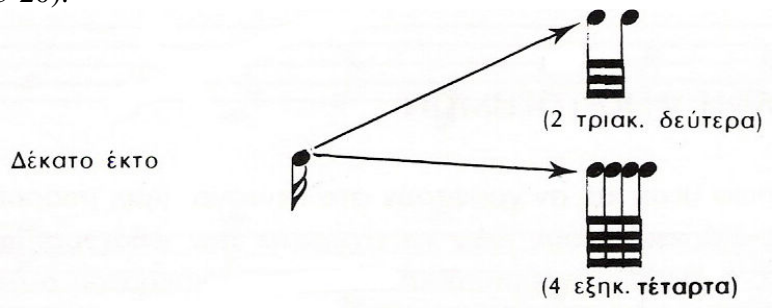
Το ένα όγδοο ισούται με 2 δέκατα έκτα ($\frac{2}{16}$), ή με 4 τριακοστά δεύτερα ($\frac{4}{32}$), ή με 8 εξηκοστά τέταρτα ($\frac{8}{64}$)(βλέπε Σχήμα 3-19).



Σχήμα 3-19: Ισοδυναμίες ογδόου.

5. Με τι ισούται το ένα δέκατο έκτο

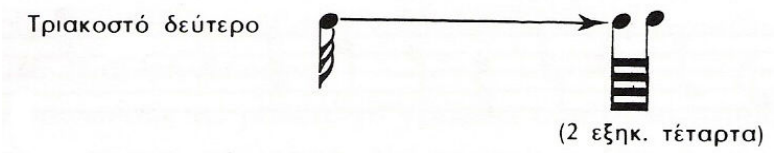
Το ένα δέκατο έκτο ισούται με 2 τριακοστά δεύτερα ($\frac{2}{32}$), ή με 4 εξηκοστά τέταρτα ($\frac{4}{64}$) (βλέπε Σχήμα 3-20).



Σχήμα 3-20: Ισοδυναμίες δεκάτου έκτου.

6. Με τι ισούται το ένα τριακοστό δεύτερο

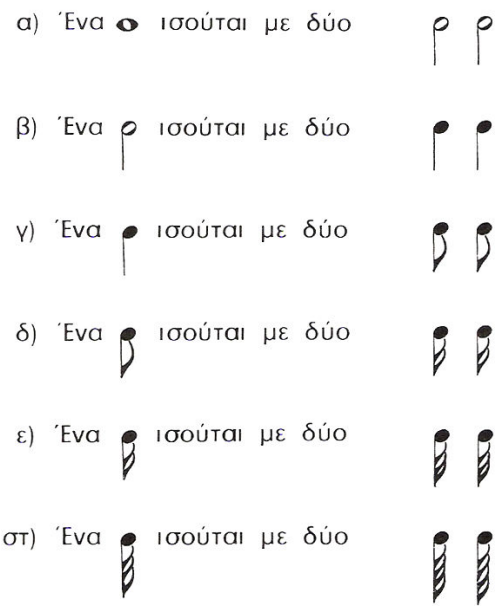
Το ένα τριακοστό δεύτερο ισούται με 2 εξηκοστά τέταρτα ($\frac{2}{64}$) όπως φαίνεται στο Σχήμα 3-21.



Σχήμα 3-21: Ισοδυναμία τριακοστού δευτέρου.

Με τι ισούται η αξία του φθογγοσήμου (από τη μεγαλύτερη στην αμέσως μικρότερη)

Από ότι είδαμε σε όλες τις διαιρέσεις αξιών, καταλήγουμε στο συμπέρασμα ότι η μεγαλύτερη αξία ισούται με δύο της αμέσως μικρότερης όπως απεικονίζεται στο Σχήμα 3.22.

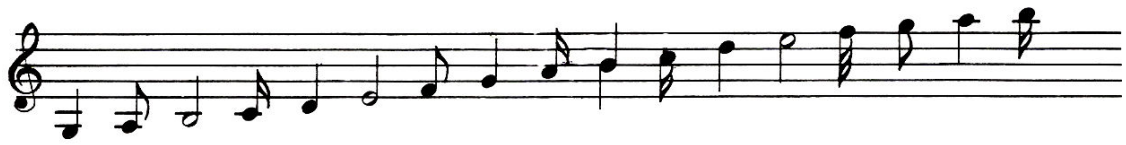


Σχήμα 3-22: Ισοδυναμία αξίας κάθε φθογγοσήμου.

Γραφή φθογγοσήμων

Οι φθόγγοι, σε όποια θέση και αν γραφτούν στο πεντάγραμμο, μπορούν να πάρουν, για τον καθορισμό της διάρκειάς τους, όλα τα σχήματα των φθογγοσήμων.
Οι ουρές των σχημάτων μισού, τετάρτου, ογδόου, δεκάτου έκτου, τριακοστού δεύτερου και εξηκοστού τετάρτου, όπως απεικονίζεται στο Σχήμα 3-23. Δηλαδή:

Οι ουρές των φθογγοσήμων, που βρίσκονται από την τρίτη γραμμή του πενταγράμμου και κάτω, γράφονται προς τα πάνω και δεξιά του φθογγοσήμου. Και στα φθογγόσημα που βρίσκοντα από την τρίτη γραμμή του πενταγράμμου και πάνω γράφονται προς τα κάτω κι αριστερά του φθογγοσήμου, αρκεί να αποτελούν μονοφωνία.



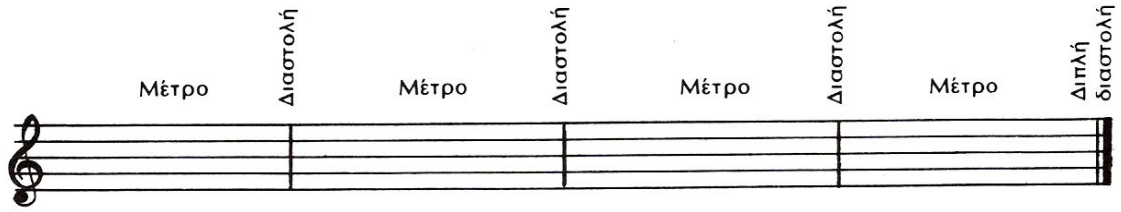
Σχήμα 3-23: Γραφή φθογγοσήμων στο πεντάγραμμο.

Μουσικό μέτρο - Διαστολή

Κάθε μουσική σύνθεση που γράφεται στο πεντάγραμμο χωρίζεται κατά διαστήματα με κάθετες γραμμές που λέγονται **διαστολές**. Η απόσταση, που χωρίζει τη μια διαστολή από την άλλη, λέγεται **μουσικό μέτρο**· αλλά και η απόσταση από το κλειδί ως τη διαστολή λέγεται και αυτή μέτρο. Το πρώτο, λοιπόν, μέτρο του πενταγράμμου γίνεται από μια διαστολή.

Μουσικό μέτρο λέγεται η απόσταση μεταξύ δύο διαστολών, και διαστολή λέγεται η κάθετη διαχωριστική γραμμή.

Όταν τελειώνει μία μουσική σύνθεση, στο τελευταίο της μέτρο η δεξιά διαστολή γράφεται διπλή, όπως φαίνεται στο Σχήμα 3-24. Η δεύτερη γραμμή της διπλής διαστολής γράφεται παχύτερη.



Σχήμα 3-24: Μουσικό μέτρο στο πεντάγραμμο.

ΣΗΜΕΙΩΣΗ: Η διπλή διαστολή μπορεί να γραφτεί και σε ένα ενδιάμεσο μέρος μιας μουσικής σύνθεσης. Αυτό γίνεται για να χωρίσει το τέλος μιας μουσικής φράσης.

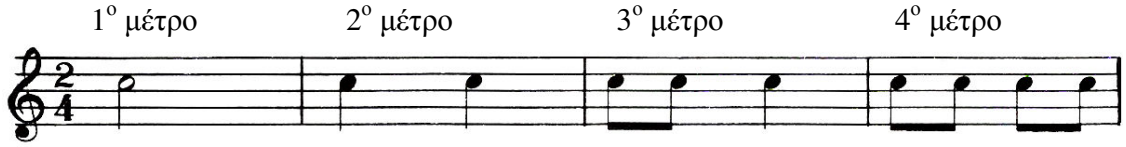
Αριθμός μέτρου

Στην αρχή κάθε μουσικής σύνθεσης, μετά το κλειδί και στα αριστερά του πρώτου μέτρου, γράφεται ένας κλασματικός αριθμός, π.χ. $\frac{2}{4}, \frac{3}{4}, \frac{4}{4}$ κ.λ.π.

Αυτός ο αριθμός δίνει: πρώτον, το όνομα του μέτρου και, δεύτερον, το ίσο άθροισμα των αξιών που θα έχει κάθε μέτρο. Αν π.χ. έχουμε στην αρχή μιας μουσικής σύνθεσης $\frac{2}{4}$ λέμε ότι η σύνθεση αυτή είναι γραμμένη σε μέτρο δύο τετάρτων και το κάθε μέτρο θα έχει

άθροισμα αξιών $\frac{2}{4}$, αν έχουμε $\frac{3}{4}$ λέμε ότι η σύνθεση είναι γραμμένη σε μέτρο τριών τετάρτων και το κάθε μέτρο θα έχει άθροισμα αξιών $\frac{3}{4}$ κ.ο.κ.

Βλέπουμε λοιπόν ότι ο κλασματικός αυτός αριθμός, επιβάλει να έχουν όλα τα μέτρα το ίδιο άθροισμα αξιών σύμφωνα με την δική του αξία.



Σχήμα 3-25: Μουσική σύνθεση σε μέτρο 2:4.

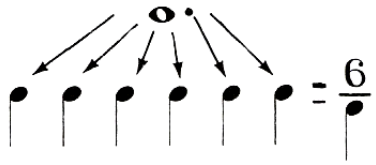
Όπως βλέπουμε στο Σχήμα 3-25, η μουσική σύνθεση είναι γραμμένη σε μέτρο $\frac{2}{4}$, και τα μέτρα της έχουν διάφορες αξίες. Κάνουμε τώρα την επαλήθευση για να δούμε μήπως υπάρχει κάποιο λάθος. Εξετάζουμε το 1ο μέτρο και βλέπουμε ότι έχει ένα μισό. Το μισό έχει $\frac{2}{4}$. Άρα είναι σωστό. Το 2ο μέτρο έχει δύο τέταρτα χωρισμένα, άρα είναι σωστό. Το 3ο μέτρο έχει 2 όγδοα και ένα τέταρτο. Τα δύο όγδοα δεν μας κάνουν ένα τέταρτο; Και ένα τέταρτο χωριστά, άρα $\frac{2}{4}$. Το 4ο μέτρο έχει 4 όγδοα. Τα δύο όγδοα μας κάνουν ένα τέταρτο. Και ένα τέταρτο που μας κάνουν τα άλλα δύο όγδοα, γίνονται $\frac{2}{4}$. Άρα και αυτό το μέτρο είναι σωστό.

Παρεστιγμένα φθογγόσημα

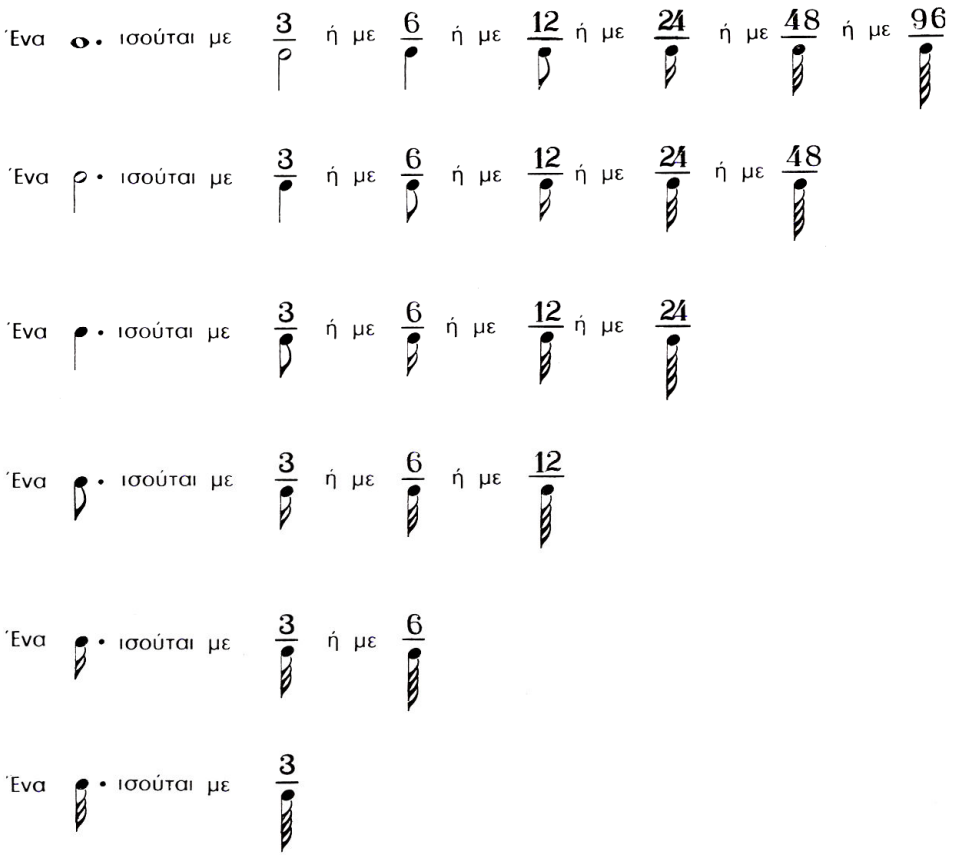
Το φθογγόσημο που έχει δεξιά του μια στιγμή (𝅗𝅥.) λέγεται παρεστιγμένο. Η στιγμή αυτή λέγεται στιγμή διαρκείας και αυξάνει τη διάρκεια του φθογγοσήμου κατά το μισό της πραγματικής του αξίας. Στο Σχήμα 3-27 απεικονίζονται οι ισοδυναμίες των παρεστιγμένων φθογγοσήμων.

Αν π.χ. έχουμε ένα φθογγόσημο σε μισό παρεστιγμένο (𝅗𝅥.) και θέλουμε να μάθουμε πόσα τέταρτα έχει, για να το βρούμε θα κάνουμε το εξής: Θα βρούμε πρώτα πόσα τέταρτα έχει η πραγματική του αξία και μετά θα προσθέσουμε από αυτό που θα βρούμε το μισό. Εδώ λοιπόν το μισό έχει πραγματική αξία $\frac{2}{4}$ και ένα τέταρτο που μας αυξάνει η στιγμή (γιατί το μισό είναι από τα $\frac{2}{4}$ το $\frac{1}{4}$) γίνεται $\frac{3}{4}$. Άρα ένα μισό παρεστιγμένο έχει $\frac{3}{4}$.

Αν τώρα θέλουμε να μάθουμε ένα ολόκληρο παρεστιγμένο (𝅝.) πόσα τέταρτα έχει θα πούμε: Το ολόκληρο χωρίς στιγμή μας κάνει $\frac{4}{4}$ και $\frac{2}{4}$ που μας αυξάνει η στιγμή σύνολο $\frac{6}{4}$, όπως φαίνεται στο Σχήμα 3-26.



Σχήμα 3-26: Ισοδυναμία ολόκληρου παρεστιγμένου.

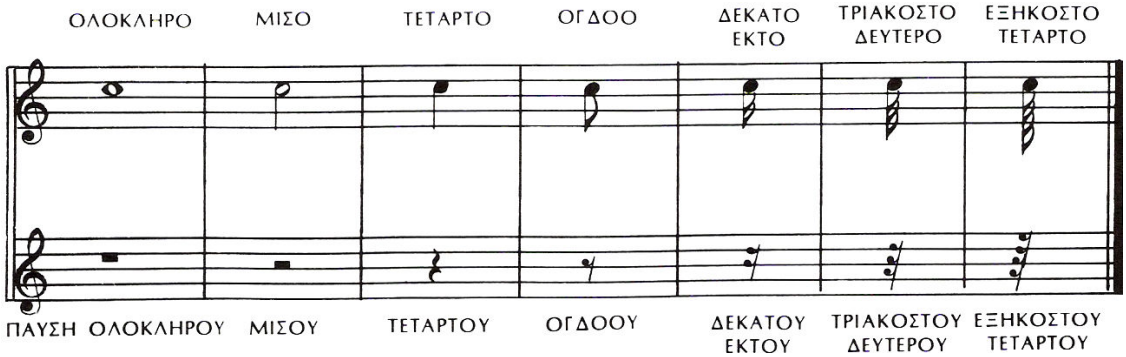


Σχήμα 3-27: Ισοδυναμίες παρεστιγμένων φθογγοσήμων.

Παύσεις

Παύσεις ονομάζουμε τα σημεία εκείνα που μας δείχνουν τον ανάλογο χρόνο διακοπής (σιωπής), στην πορεία ενός μουσικού κομματιού. Όσα σχήματα έχουμε για τις διάρκειες των φθογγοσήμων άλλα τόσα έχουμε και για τις διάρκειες των παύσεων. Η διαφορά είναι ότι τα σχήματα των παύσεων μας καθορίζουν διάρκεια διακοπής χωρίς ήχο, ενώ τα σχήματα των φθογγοσήμων μας καθορίζουν διάρκεια με ήχο.

Κάθε διάρκεια φθογγοσήμου έχει και την αντίστοιχη παύση της. Στο Σχήμα 3-28 παρουσιάζονται αναλυτικά τα σχήματα των παύσεων μαζί με τα σχήματα διαρκείας των φθογγοσήμων.



Σχήμα 3-28: Πίνακας σχημάτων φθογγοσήμων και παύσεων.

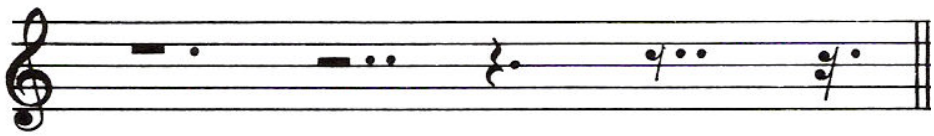
Η παύση ολόκληρου και η παύση μισού κατά το σχήμα είναι ίδιες με τη διαφορά ότι η παύση του ολόκληρου γράφεται κάτω ακριβώς από την τέταρτη γραμμή του πενταγράμμου, ενώ του μισού πάνω ακριβώς από την τρίτη γραμμή με μια παύλα.

Οι άλλες παύσεις δεν έχουν ορισμένη θέση για το γράψιμό τους στο πεντάγραμμο. Ανάλογα με τη θέση που έχουν τα φθογγόσημα πριν ή μετά την παύση, γράφεται και η παύση στο ανάλογο ύψος του πενταγράμμου.

Όπως κάθε αξία φθογγοσήμου υποδιαιρείται σε 2 ή και περισσότερες αξίες, το ίδιο γίνεται και με τις αξίες των παύσεων.

Π.χ. μια παύση ολόκληρου είναι ίση με 2 παύσεις μισού ή με 4 παύσεις τετάρτου ή με 8 παύσεις ογδόου ή με 16 παύσεις δεκάτου έκτου κ.ο.κ.

Οι παύσεις επίσης μπορούν να γραφούν παρεστιγμένες και δις παρεστιγμένες όπως και τα φθογγόσημα όπως στο Σχήμα 3-29.



Σχήμα 3-29: Παρεστιγμένες και δις παρεστιγμένες παύσεις.

Σημεία αλλοίωσης

Σημεία αλλοίωσης λέγονται τα σημεία αυτά που γράφονται αριστερά των φθόγγων και μεταβάλλουν (αλλοιώνουν) το ύψος τους. Τα σημεία αυτά είναι απλά και διπλά.

Απλά σημεία αλλοίωσης

Τα απλά σημεία αλλοίωσης ανεβάζουν ή κατεβάζουν το ύψος των φθόγγων κατά ένα ημιτόνιο (μισή φωνή) και είναι:

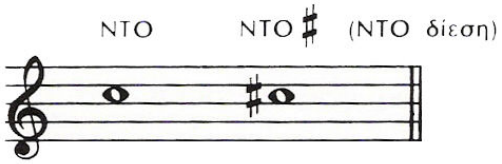
α) **η δίεση (#)** που ανεβάζει το φθόγγο ένα ημιτόνιο,
β) **η ύφεση ή μπεμόλ** (♭) που κατεβάσει το φθόγγο ένα ημιτόνιο και
γ) **η αναίρεση** (♮) που επαναφέρει ένα αλλοιωμένο φθόγγο στο φυσικό του ύψος.

Διπλά σημεία αλλοίωσης

Τα διπλά σημεία αλλοίωσης ανεβάζουν ή κατεβάζουν το ύψος των φθόγγων κατά δύο ημιτόνια, δηλαδή ένα τόνο και είναι:

α) **η διπλή δίεση (** 𝄪 **)** που ανεβάζει το φθόγγο κατά δύο ημιτόνια (ένα τόνο),
β) **η διπλή ύφεση** (𝄫) που κατεβάζει το φθόγγο κατά δύο ημιτόνια (ένα τόνο) και
γ) **η διπλή αναίρεση** (♮♮) που επαναφέρει ένα αλλοιωμένο φθόγγο με διπλή δίεση ή διπλή ύφεση, στο φυσικό του ύψος.
(Η διπλή αναίρεση σπάνια χρησιμοποιείται).

Για να ονομάσουμε ένα φθόγγο με δίεση, με ύφεση ή αναίρεση πρέπει το σημείο αλλοίωσης να βρίσκεται στα αριστερά του φθόγγου, όπως απεικονίζεται στο Σχήμα 3-30.

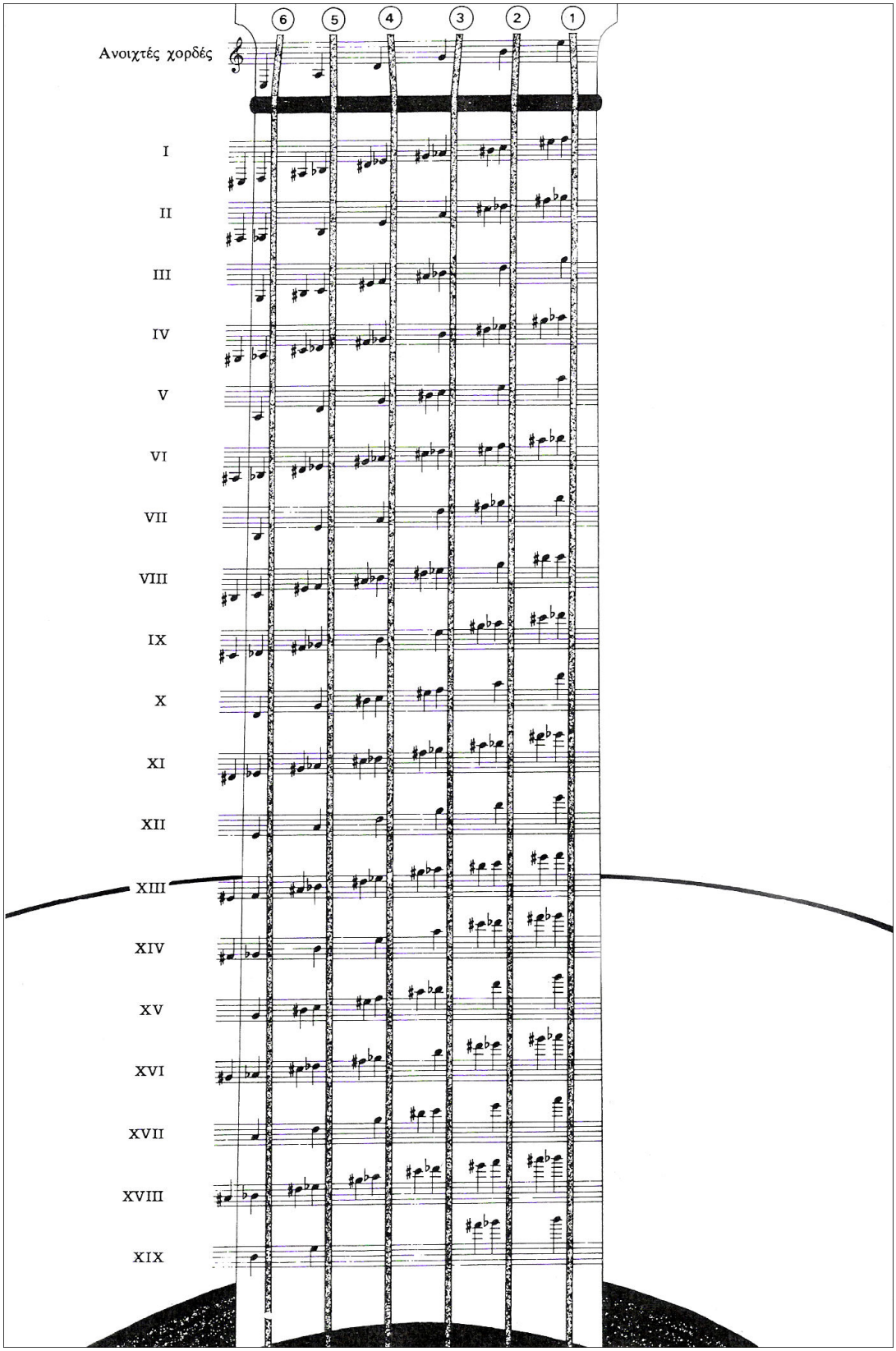


Σχήμα 3-30: Απεικόνιση φθογγοσήμου ΝΤΟ δίεση στο πεντάγραμμο.

Η δίεση ανήκει στο δεύτερο ΝΤΟ. Για να προφέρουμε τώρα αυτούς τους φθόγγους θα πούμε: ΝΤΟ, ΝΤΟ#. Όπως βλέπουμε για να πούμε το φθόγγο με τη δίεση, λέμε πρώτα το φθόγγο και μετά τη δίεση και ας είναι γραμμένη πρώτα η δίεση, δηλαδή δε λέμε ποτέ δίεση ΝΤΟ αλλά ΝΤΟ δίεση. Το ίδιο γίνεται και για την ύφεση, την αναίρεση καθώς και για τα διπλά σημεία αλλοίωσης.

Αντιστοιχία φθογγοσήμων στην ταστιέρα της κιθάρας

Στο Σχήμα 3-31 παρουσιάζεται η αντιστοιχία των φθογγοσήμων (νοτών) του πενταγράμμου στις θέσεις της ταστιέρας (μπράτσου) της κιθάρας. Συγκεκριμένα απεικονίζεται η οπτική αναπαράσταση της ταστιέρας της κιθάρας με τις χορδές, τα τάστα και τα φθογγόσημα που αντιστοιχούν σε κάθε τάστο της χορδής.



Σχήμα 3-31: Αντιστοιχία φθογγοσήμων στην ταστιέρα της κιθάρας.

Τρόπος ανάγνωσης του Chord - Lexicon

Ονοματολογία: Το αρχικό λατινικό στοιχείο ενός ακκόρντου αντιπροσωπεύει την ονοματολογία του, όπως απεικονίζεται στον Πίνακα 3-1.

A	:	Λα	**E**	:	Μι
B	:	Σι	**F**	:	Φα
C	:	Ντο	**G**	:	Σολ
D	:	Ρε			

Πίνακας 3-1: Διεθνής ονοματολογία ακκόρντων.

Τα στοιχεία που παραθέτονται στον Πίνακα 3-2 αντιπροσωπεύουν το είδος του:

A	:	Λα (ματζόρε) (όταν η ονοματολογία του ακκόρντου δεν συνοδεύεται από άλλη ένδειξη εννοείται ότι το ακκόρντο είναι ματζόρε).
Am	:	Λα μινόρε (minor).
A maj7	:	Λα ματζόρε μεγάλης εβδόμης (major 7).
Am maj 7	:	Λα μινόρε μεγάλης εβδόμης.
A 7	:	Λα (ματζόρε) εβδόμης (μικρής).
Am 7	:	Λα μινόρε εβδόμης.
A 6	:	Λα (ματζόρε) έκτης.
Am 6	:	Λα μινόρε έκτης.
A 9	:	Λα (ματζόρε) ενάτης.
Am 9	:	Λα μινόρε ενάτης.
A $^{6}_{9}$	:	Λα (ματζόρε) έκτης - ενάτης.
A 11	:	Λα (ματζόρε) ενδεκάτης.
A 13	:	Λα (ματζόρε) δεκάτης τρίτης.
A $^{+}$	:	Λα αυξημένη.
A +7	:	Λα αυξημένη με εβδόμη.
A $^{\oslash}$ **7**	:	Λα ελαττωμένη με εβδόμη.
A°7	:	Λα ελαττωμένη με εβδόμη ελαττωμένη (diminuita).
A#9	:	Λα (ματζόρε) με μεγάλη ενάτη.
A sus4	:	Λα (ματζόρε) με τετάρτη (suspended 4^{th} - πρόσθετη $4^{η}$).
A 7sus4	:	Λα (ματζόρε) εβδόμης με τετάρτη.

Πίνακας 3-2: Είδη ακκόρντων.

Τα ακκόρντα που δίνονται στο CHORD – LEXICON μπορεί σε κάποιες εκδόσεις να τα συναντήσετε με διαφορετική σημειογραφία, για το λόγο αυτό παραθέτουμε στον Πίνακα 3-3 τις γνωστότερες αντιστοιχίες.

A maj 7	**AM 7**	**A#7**	
A ⦸7	**Am 7 ($_b$5)**	**Am 7(-5)**	**A dim**
A° 7	**A°**	**A dim**	
A +7	**A 7^+**	**A 7 (#5)**	**A 7(+5)**
A 9	**A add9**	**A 7 (add9)**	
A $^6_9$	**A 6 (add9)**		
Am	**A 7 (add11)**		
A $^+$	**A aug**	**A #5**	**A +5**
A$_b$5	**A (dim5)**	**A (-5)**	

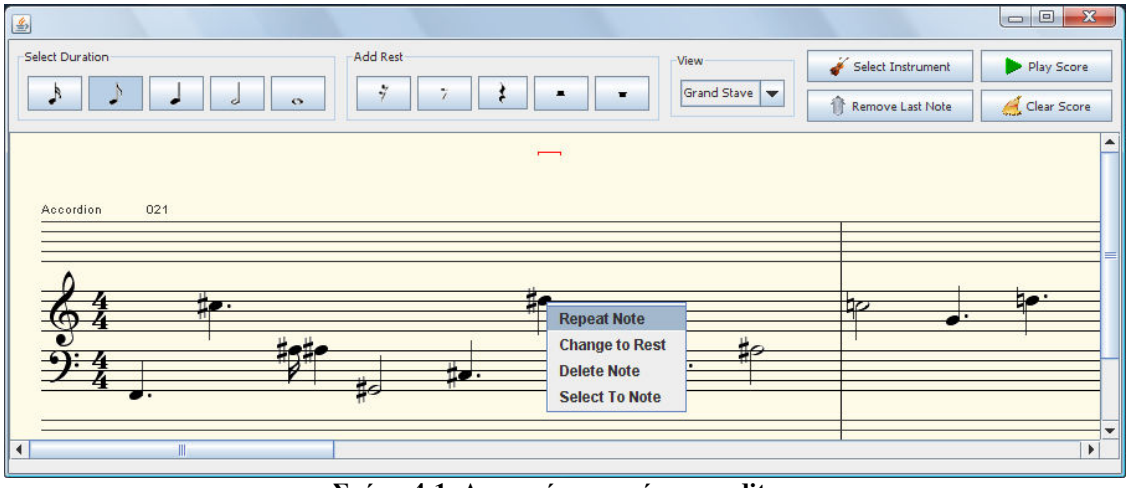
Πίνακας 3-3: Σημειογραφίες ακκόρντων.

4. Προδιαγραφές πτυχιακής εργασίας

Η πτυχιακή αυτή εργασία έχει ως στόχο την δημιουργία μιας ηλεκτρονικής μουσικής κοινότητας για την εκμάθηση κιθάρας. Έχοντας ως γνώμονα την εύκολη και γρήγορη εκμάθηση της κιθάρας ακόμα και από έναν αρχάριο μαθητή, καταλήξαμε στη δημιουργία μιας διεπάφης που θα παρέχει στο χρήστη οπτική απεικόνιση των νοτών που υπάρχουν στο πεντάγραμο στην ταστιέρα της κιθάρας. Με αυτό τον τρόπο πετύχαμε την γρήγορη εξοικείωση του μαθητή με τις νότες του πενταγράμμου και την αντιστοιχία τους στην κιθάρα.
Συγκεκριμένα ο score editor όπως μας δόθηκε παρείχε την εξής λειτουργικότητα στο χρήστη:

1. Σύνθεση σε μουσικό πεντάγραμμο
2. Επιλογή μουσικού οργάνου από τη λίστα μουσικών οργάνων που παρέχει η βιβλιοθήκη JMusic
3. Εναλλαγή παρτιτούρας
4. Επιλογή νότας/ων από το πεντάγραμο(multiple selection) με κόκκινη ένδειξη της επιλογής στο πάνω μέρος της παρτιτούρας
5. Προσθήκη / Διαγραφή / Μετακίνηση νότας στην παρτιτούρα
6. Καθαρισμό παρτιτούρας
7. Αναπαραγωγή παρτιτούρας

Αν και ο score editor φαίνεται να καλύπτει ένα ευρύ φάσμα λειτουργιών, στην πράξη υπήρχαν αδυναμίες. Όταν τρέξαμε την εφαρμογή για πρώτη φορά διαπιστώσαμε ότι είχε bugs. Επίσης, ο σχεδιασμός και η επιλογή των components που συνθέτουν την διεπαφή του score editor που παρουσιάζεται στο Σχήμα 4-1, δεν ήταν ο καλύτερος για την λειτουργικότητα που παρείχε η εφαρμογή.



Σχήμα 4-1: Διεπαφή αρχικού score editor.

Μετά από λεπτομερή μελέτη του κώδικα που μας δόθηκε, αποφασίσαμε να διορθώσουμε τις αδυναμίες που υπήρχαν, και να βελτιώσουμε τον κώδικα της εφαρμογής ώστε να παρέχει την ίδια λειτουργικότητα, μειώνοντας παράλληλα τις γραμμές του κώδικα.
Έτσι μετά την ολοκλήρωση της πτυχιακής αυτής εργασίας η εφαρμογή που θα έχει υλοποιηθεί θα δίνει στο χρήστη τις εξής δυνατότητες:

1. Σύνθεση σε μουσικό πεντάγραμμο
2. Επιλογή μουσικού οργάνου από λίστα μουσικών οργάνων που ανήκουν στην κατηγορία «κιθάρα»
3. Εστίαση στο πάνω και το κάτω μέρος της παρτιτούρας με εκάστοτε επιλογή από το χρήστη (εναλλαγή παρτιτούρας σε Treble / Bass)
4. Επιλογή νότας από το πεντάγραμο με ταυτόχρονη αλλαγή του χρώματος της επιλεγμένης νότας (κόκκινο) και εμφάνιση των θέσεων της νότας στην ταστιέρα της κιθάρας
5. Προσθήκη / Διαγραφή / Μετακίνηση νότας στην παρτιτούρα
6. Χρήση μετρονόμου
7. Καθαρισμό παρτιτούρας
8. Αναπαραγωγή παρτιτούρας
9. Αποθήκευση παρτιτούρας

Η διεπαφή της εφαρμογής θα περιέχει μενού File όπου μέσα θα υπάρχει ‘Clear Stave’, ‘Save Stave’, ‘Play Stave’ και ‘Exit’. Θα σχεδιαστεί μενού Tools όπου μέσα σε αυτό θα τοποθετηθούν εργαλεία όπως ο Μετρονόμος καθώς και μενού Help.

Μόλις τρέχει η εφαρμογή θα δημιουργείται ένα κεντρικό Panel που θα αποτελείται από δύο Panels.Το πάνω θα είναι ο editor (η παρτιτούρα) και από κάτω θα είναι το Panel με την ταστιέρα της κιθάρας.

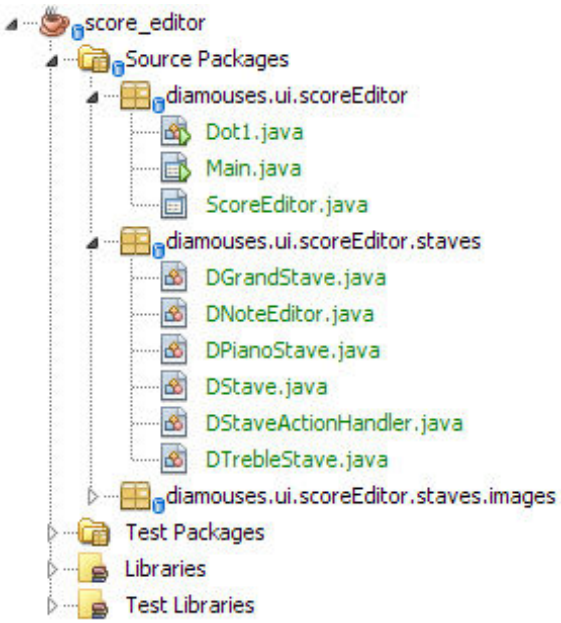
Τέλος, η εφαρμογή θα ενσωματωθεί σε ιστοσελίδα που θα καλύπτει τις ανάγκες μιας ηλεκτρονικής κοινότητας που αφορά την εκμάθηση κιθάρας. Συγκεκριμένα θα παρέχει τις εξής δυνατότητες στο χρήστη:

- Εγγραφή στην σελίδα
- Ηλεκτρονική τάξη όπου ο χρήστης θα πειραματίζεται με την εφαρμογή
- Λήψη αρχείων θεωρίας και ασκήσεων που έχει ανεβάσει ο καθηγητής
- Φόρουμ, για συζήτηση με μαθητές και καθηγητές
- Βοήθεια με οδηγίες χρήσης της εφαρμογής

Η ιστοσελίδα θα σχεδιαστεί ώστε μόνο οι εγγεγραμένοι χρήστες (μαθητές ή καθηγητές) να έχουν πρόσβαση σε όλες τις ενότητες της ιστοσελίδας.

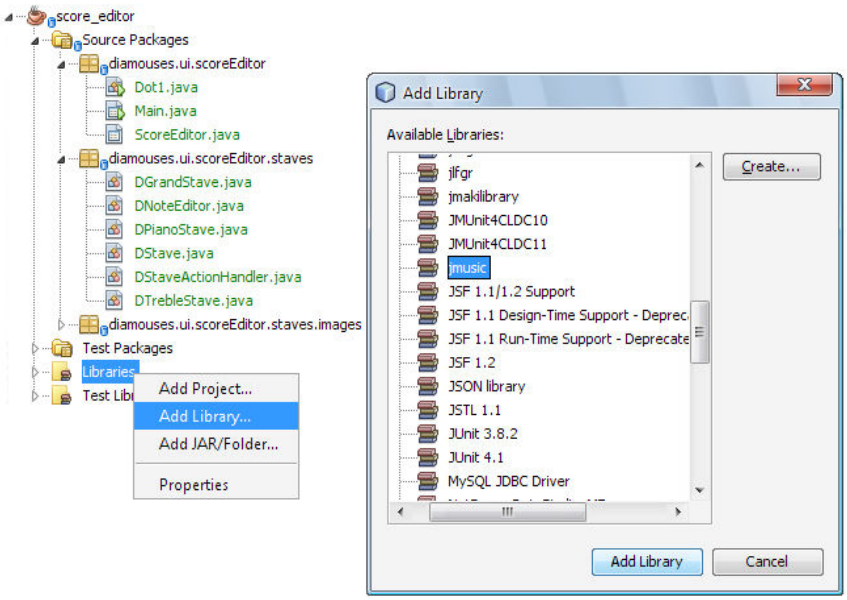
5. Ανάλυση του κώδικα score-editor

Ο score editor που μας δόθηκε αποτελείται από εννιά αρχεία Java, ένα φάκελο που περιέχει τις εικόνες της εφαρμογής, δύο αρχεία τύπου form και τη βιβλιοθήκη JMusic.jar που βρίσκεται στο φάκελο Libraries. Στο Σχήμα 5-1 παρουσιάζεται η δομή των αρχείων του score editor.



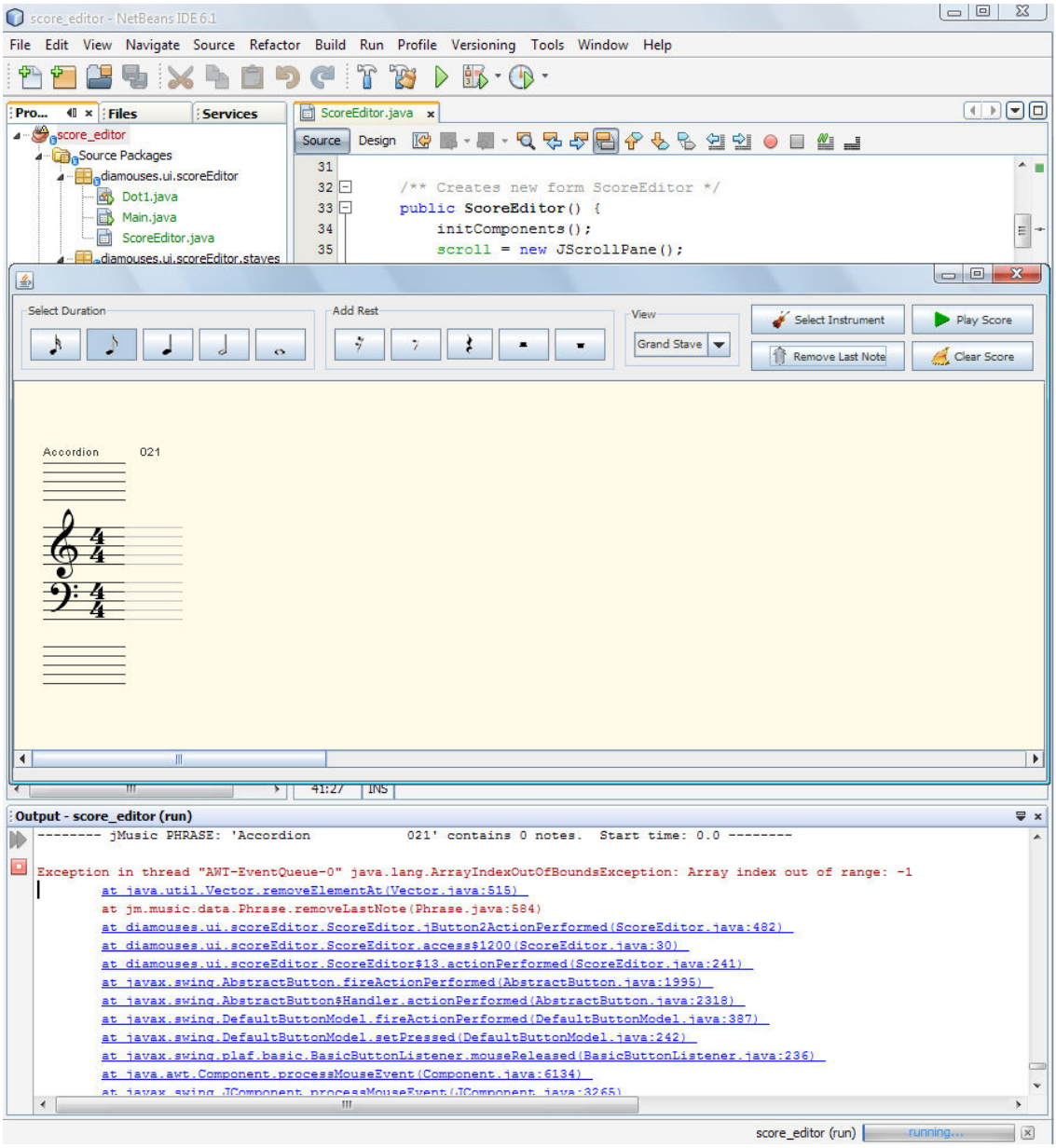
Σχήμα 5-1: Δομή αρχείων του score editor.

Είναι απαραίτητο πριν τρέξουμε την εφαρμογή για πρώτη φορά να ορίσουμε την ύπαρξη της βιβλιοθήκης JMusic.jar όπως φαίνεται στο Σχήμα 5-2.



Σχήμα 5-2: Προσθήκη βιβλιοθήκης JMusic στον score editor.

Αφού τρέξουμε το πρόγραμμα θα παρατηρήσουμε σταδιακά κάποιες αδυναμίες του. Το πιο προφανές bug της εφαρμογής που μάλιστα πετάει και exceptions(βλέπε Σχήμα 5-2) εμφανίζεται όταν έχουν αφαιρεθεί όλες οι νότες και ο χρήστης πατήσει το κουμπί Remove Last Note.



Σχήμα 5-3: Exception του score editor.

Μια εύκολη λύση στο παραπάνω πρόβλημα είναι να γίνεται disabled το κουμπί ‘Remove Last Note’ όταν δεν υπάρχουν νότες στο πεντάγραμμο. Ας αφήσουμε όμως τις αδυναμίες της εφαρμογής και ας μελετήσουμε τον τρόπο με τον οποίο σχεδιάστηκε. Ας δούμε κάθε αρχείο Java ξεχωριστά.

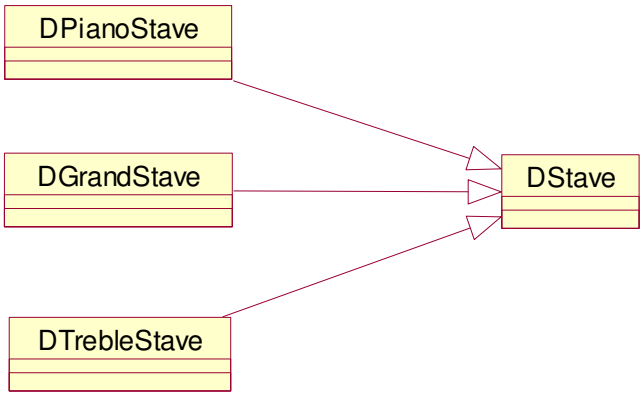
5.1 Αρχείο: Main.java

Εδώ βρίσκεται ο κώδικας που είναι απαραίτητος για την εκκίνηση της εφαρμογής. Η κλάση Main είναι τύπου JFrame και στη main μέθοδο, την μέθοδο που καλείται όταν τρέχει η κλάση αυτή καλείται ο constructor της Main, ο οποίος αρχικοποιεί τον ScoreEditor και κατόπιν τον τοποθετεί στην οθόνη του χρήστη.

```
public class Main extends javax.swing.JFrame {

    /** Creates new form Main */
    public Main() {
        initComponents();
        ScoreEditor score = new ScoreEditor();
        getContentPane().add(score, BorderLayout.CENTER);
        pack();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this
method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code
">
    private void initComponents() {


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE
);
        pack();
    }// </editor-fold>

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Main().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    // End of variables declaration

}
```

5.2 Αρχεία: DPianoStave,Dstave,DGrandStave,DTrebleStave

Ας δούμε τα επόμενα 4 αρχεία με χρήση ενός λογισμικού που δημιουργεί και παρουσιάζει UML Diagrams [1] το Rational Rose [23].



Σχήμα 5-4: Uml διάγραμμα κλάσεων των staves του score editor.

Τα παραπάνω 4 αρχεία παρουσιάζονται μαζί για να εξηγηθεί η τεχνοτροπία και ο αρχικός σχεδιασμός τους. Παρατηρούμε ότι υπάρχουν 3 διαφορετικές παρτιτούρες. Η πρώτη για πιάνο, η δεύτερη για stave και η τρίτη για τρέμπλο. Οι τρεις παραπάνω κλάσεις (βλέπε Σχήμα 5-4) που αναπαριστούν παρτιτούρες υλοποιούν το αντικείμενο DStave.

Εδώ πρέπει να παρουσιάσουμε μια από τις ιδιότητες της Java καθώς και όλων των αντικειμενοστραφών γλωσσών προγραμματισμού. Η ιδιότητα αυτή ονομάζεται κληρονομικότητα [24] ή inheritance στα αγγλικά.

Η ιδιότητα αυτή μπορεί να περιγραφεί με ένα απλό παράδειγμα. Ας υποθέσουμε ότι θέλουμε να περιγράψουμε όλους τους πολιτισμούς του γαλαξία μας και ότι υπάρχουν άνθρωποι, αρειανοί και δεκάδες άλλοι πολιτισμοί. Στην αντικειμενοστραφή γλώσσα πρέπει να δημιουργήσουμε ένα αντικείμενο για κάθε είδος πολιτισμού. Μπορούμε εύκολα να παρατηρήσουμε ότι όλα τα όντα έχουν κάποια κοινά χαρακτηριστικά, όπως όνομα, ηλικία και φύλλο. Αντί λοιπόν στο κάθε αντικείμενο να έχουμε τον παρακάτω κώδικα:

```
// Variables
int age;
String name;
String sex;

// Setters & Getters
public int getAge() {
        return age;
}
public void setAge(int age) {
        this.age = age;
```

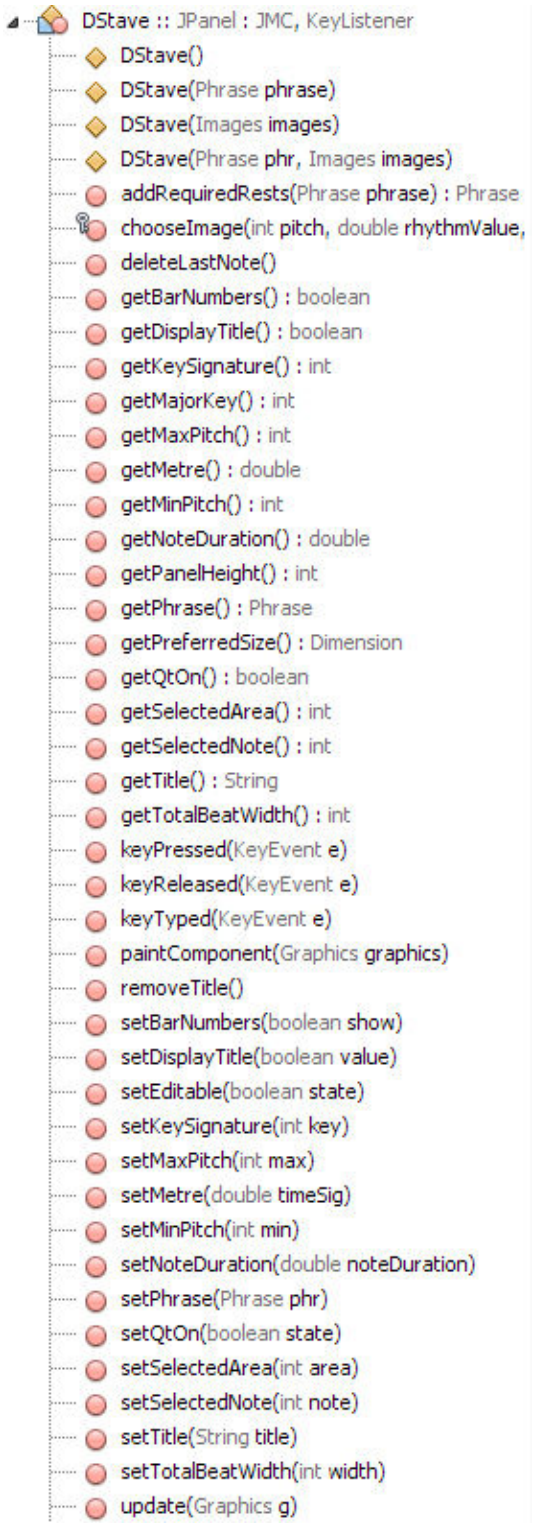
```
}
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public String getSex() {
        return sex;
}
public void setSex(String sex) {
        this.sex = sex;
}
```

Τον τοποθετούμε σε ένα αντικείμενο το οποίο ονομάζουμε On.java για παράδειγμα. Για κάθε πολιτισμό χρειάζεται ένα αντικείμενο. Το κάθε αντικείμενο κάνει extend το αντικείμενο «μπαμπά» (το parent object On.java) και «κληρονομεί» τον κώδικα που παρουσιάσαμε παραπάνω.

Έτσι αποφεύγεται η ύπαρξη του ίδιου κώδικα σε 10αδες ή ακόμα και εκατοντάδες αρχεία-αντικείμενα Java.

Το χαρακτηριστικό της κληρονομικότητας έχει χρησιμοποιηθεί και στον αρχικό score-editor. Τα κοινά χαρακτηριστικά όλων των παρτιτούρων έχουν τοποθετηθεί στο αρχείο DStave.java και κάθε ιδιαίτερη παρτιτούρα κάνει extend το αρχείο αυτό και κληρονομεί τον κώδικα του.

Έτσι στο DStave.java έχουμε τα ακόλουθα attributes τα οποία είναι κοινά για όλα τα πεντάγραμμα και απεικονίζονται στο Σχήμα 5-5.



Σχήμα 5-5: Attributes του DStave.java.

Το Dstave είναι ένα αρκετά περίπλοκο αντικείμενο με 650 γραμμές κώδικα, δεκάδες μεταβλητές και μεθόδους.

Ένα από τα χαρακτηριστικά του είναι ότι περιέχει 33 μεταβλητές τύπου java.awt.Image.

Περιέχει μεταβλητές για τα offset, τα margin, το μέτρο, το κλειδί, το τίτλο, μεθόδους για το pitch, το beat για προσθήκη και αφαίρεση παύσεων, για διαγραφή της τελευταίας νότας και άλλες χρήσιμες μεταβλητές και μεθόδους.

Πρέπει να τονισθεί μια από τις μεταβλητές που περιέχει και είναι τύπου `jm.music.data.Phrase`. Μέσα σε αυτή τη μεταβλητή τοποθετείται το `JMusicString`, ένα String στο notation της βιβλιοθήκης `JMusic` που περιέχει τα μουσικά δεδομένα.

Ένας αποτελεσματικότερος τρόπος σχεδίασης του DStave θα ήταν τοποθετώντας σε άλλα αντικείμενα μέρος του κώδικα ώστε να υπάρχουν περισσότερα, αλλά μικρότερα σε όγκο αντικείμενα, όπου το καθένα θα περιέχει συγκεκριμένη δόμη και χρήση.

Μελετώντας τα child-objects παρατηρούμε ότι το κάθε ένα περιέχει περίπου 300 γραμμές κώδικα εκ των οποίων οι 250 βρίσκονται στη μέθοδο `paint` και αφορούν την εμφάνιση της παρτιτούρας στην οθόνη, και την αντίστοιχη θέση του κάθε μουσικού συμβόλου.

Τέλος, να σχολιαστεί ότι στην εφαρμογή ενώ το JComboBox δείχνει 4 παρτιτούρες (βλέπε Σχήμα 5-6)

Σχήμα 5-6: Απεικόνιση του JComboBox του score editor.

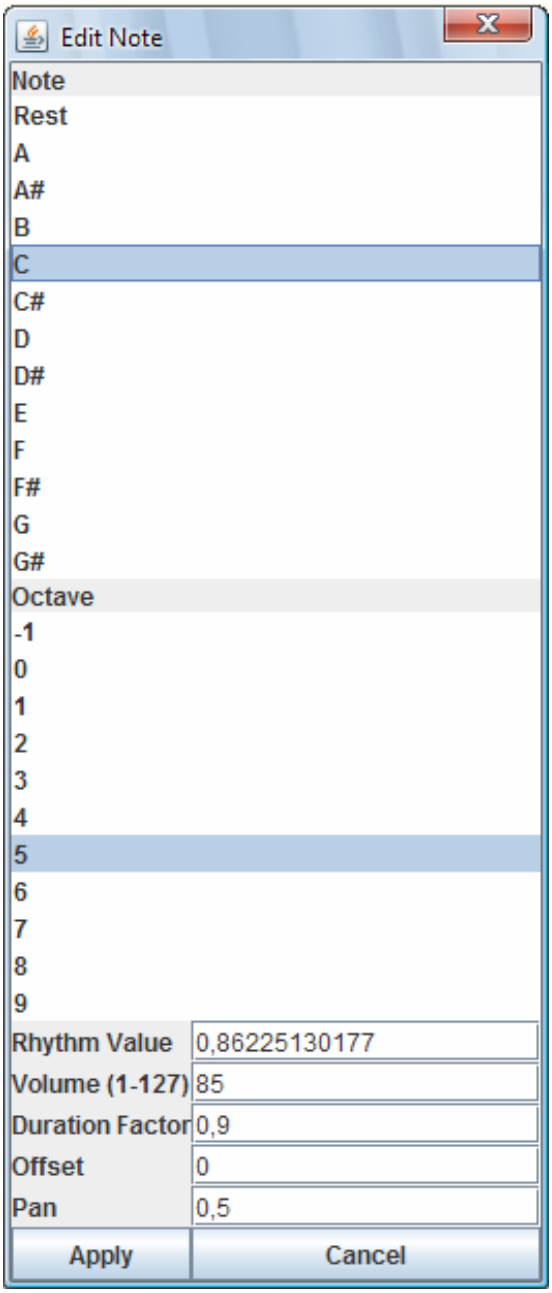
Στην πράξη υπάρχουν μόνο 3 διαφορετικές παρτιτούρες.

```
if(item.equalsIgnoreCase("Grand Stave")){
    scorePanel.remove(stave);
    stave = new DGrandStave(phr);
    scorePanel.add(stave,BorderLayout.CENTER);
}else if(item.equalsIgnoreCase("Piano Stave")){
    scorePanel.remove(stave);
    stave = new DPianoStave(phr);
    scorePanel.add(stave,BorderLayout.CENTER);
}else if(item.equalsIgnoreCase("Treble Stave")){
    scorePanel.remove(stave);
    stave = new DTrebleStave(phr);
    scorePanel.add(stave,BorderLayout.CENTER);
}else if(item.equalsIgnoreCase("Bass Stave")){
    scorePanel.remove(stave);
    stave = new DPianoStave(phr);
    scorePanel.add(stave,BorderLayout.CENTER);
}
```

Το **DPianoStave.java** χρησιμοποιείται και στο *'Bass Stave' και στο 'Piano Stave'*.

5.3 Αρχεία: DNoteEditor και DStaveActionHandler

Ξεκινώντας από το αρχείο `DNoteEditor.java` παρατηρούμε ότι δε χρησιμοποιήθηκε ποτέ στον score-editor, καθώς είναι commented out ο κώδικας που εμφανίζει το control που ανοίγει το παράθυρο που απεικονίζεται στο Σχήμα 5-7. Για να εμφανιστεί το παράθυρο χρειάστηκε να κάνουμε uncomment ορισμένες γραμμές κώδικα στο DStaveActionHandler.



Σχήμα 5-7: Παράθυρο επεξεργασίας των παραμέτρων μιας νότας.

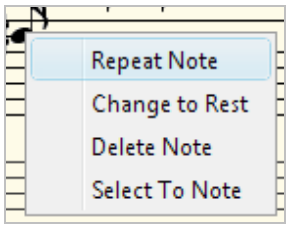
Αυτό που παρατηρούμε στο DNoteEditor.java είναι ότι εμφανίζει στοιχεία ώστε ο χρήστης να κάνει edit τις παραμέτρους μιας νότας και συγκεκριμένα:
Να αλλάξει τη μουσική νότα
Να τροποποιήσει την τιμή του ρυθμού της νότας
Να αλλάξει την ένταση με την οποία θα παίξει η νότα
Να αλλάξει το duration της
Να αλλάξει το offset
Να αλλάξει το pan

Ο κώδικας αυτού του αρχείου είναι μέρος της βιβλιοθήκης JMusic.

Ο κώδικας που έπρεπε να γίνει uncomment για να εμφανιστεί ο `DNoteEditor` ακολουθεί.

```
    // constructor
    DStaveActionHandler(DStave stave) {
        theApp = stave;

        noteContextMenu = new JPopupMenu();

        // ---------------------------
        // WE DONT NEED AN EDIT BUTTON
        // ---------------------------
//        editNote = new JMenuItem("Edit Note");
//        editNote.addActionListener(this);
//        noteContextMenu.add(editNote );
```

Το `DStaveActionHandler.java` χρησιμοποιείται από το parent-class `DStave.java` και αποτελεί τον `ActionHandler` της παρτιτούρας. Περιέχει τον κώδικα που κάνει control την εφαρμογή, δηλαδή χειρίζεται τα mouse-clicks, υλοποιεί το drag & drop και περιέχει κώδικα για τα κουμπιά και τις επιλογές της εφαρμογής.
Επίσης περιέχει κώδικα που εμφανίζει το Popup menu που φαίνεται στο Σχήμα 5-8.

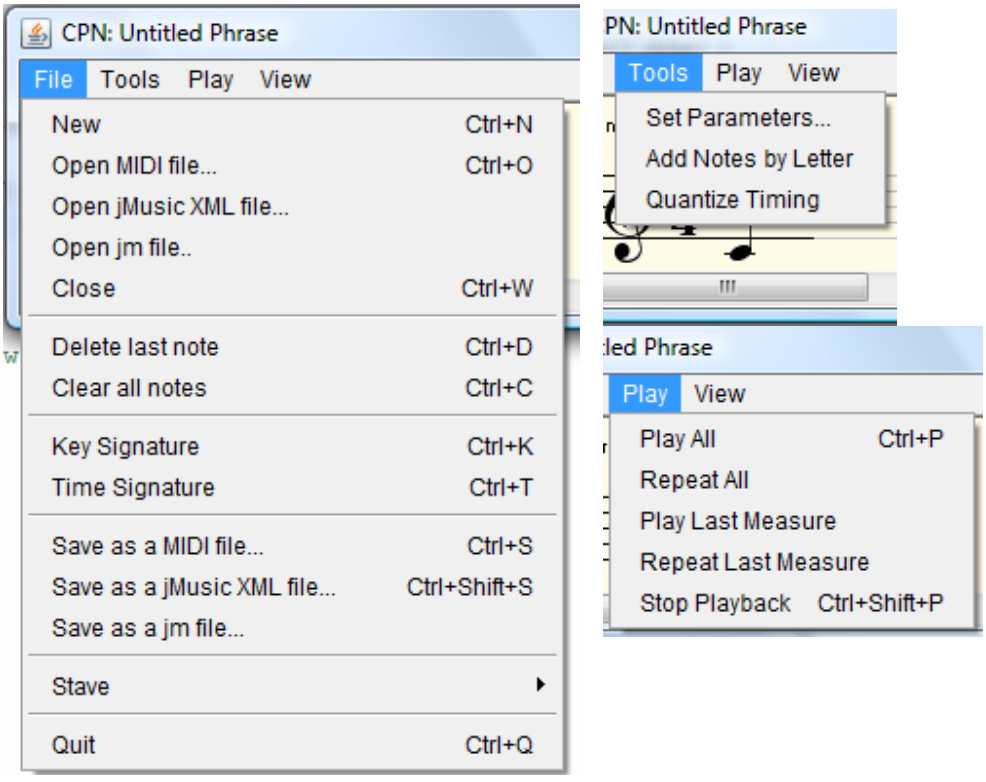


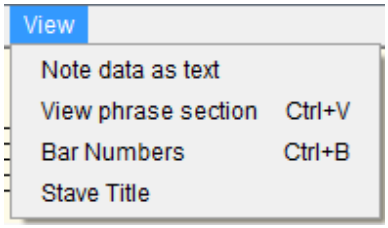
Σχήμα 5-8: Popup menu του score editor.

5.4 Αρχεία: Dot1.java, Main.form, ScoreEditor.form

Το αρχείο Dot1.java περιέχει κώδικα που καλεί το View.notate(), μια μέθοδο της JMusic βιβλιοθήκης που παρουσιάζει στο χρήστη μια φράση του JMusic notation και το ολοκληρωμένο γραφικό περιβάλλον που παρέχει η κλάση View το οποίο απεικονίζεται στο Σχήμα 5-9.

```
package diamouses.ui.scoreEditor;

import jm.music.data.*;
import jm.JMC;
import jm.util.*;

public class Dot1 implements JMC {
   public static void main(String[] args) {
      Note n;
      n = new Note(C4, QUARTER_NOTE);
      Phrase phr = new Phrase();
      phr.addNote(n);
      //View.sketch(phr);
      View.notate(phr);
      //View.histogram(new Score(new Part(phr)));
   }
}
```



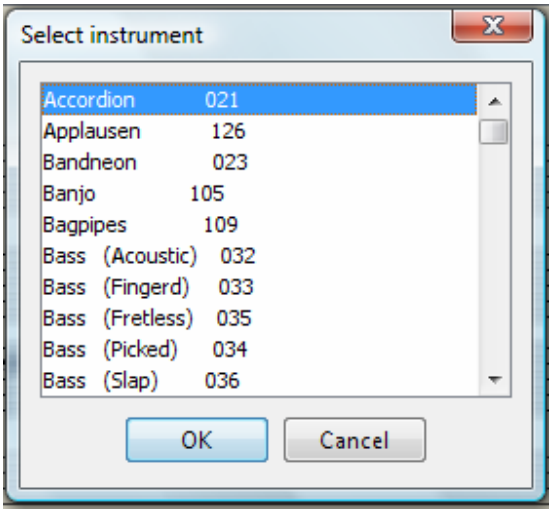


Σχήμα 5-9: Γραφικό περιβάλλον κλάσης View.

Τα δύο αρχεία τύπου .form περιέχουν User Interface δεδομένα, όπως αυτά σχεδιάστηκαν από το γραφικό περιβάλλον του εργαλείου Netbeans.

5.5 Αρχείο: ScoreEditor.java

Το αρχείο ScoreEditor αν και περιέχει 600 γραμμές κώδικα μπορεί εύκολα να κατανοηθεί. Περιέχει τον κώδικα που «χτίζει» το γραφικό περιβάλλον, εμφανίζει την επιλογή των μουσικών οργάνων όταν ο χρήστης επιλέγει 'Select Instrument' (βλέπε Σχήμα 5-10).

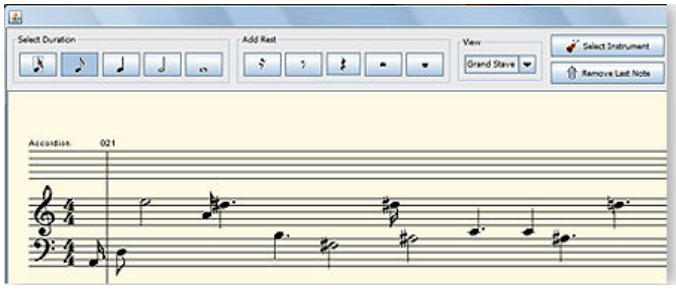


Σχήμα 5-10: Παράθυρο επιλογής μουσικού οργάνου.

Επίσης περιέχει control κώδικα για όλα τα κουμπιά που εμφανίζονται στο πάνω μέρος της εφαρμογής (βλέπε Σχήμα 5-11).

Σχήμα 5-11: Απεικόνιση κουμπιών του score editor.

Τέλος, περιέχει κώδικα που τοποθετεί 40 τυχαίες νότες στην παρτιτούρα όπως φαίνεται στο Σχήμα 5-12.



```
Vector<Note> notes = new Vector<Note>();
 for(int i=0;i<40;i++){
   double min=41;
   double max=78;
   double pitch = (max-min)*Math.random()+min;
   Note note1 = new Note((int)pitch, 2*Math.random());
   notes.add(note1);
```

Σχήμα 5-12: Κώδικας εισαγωγής τυχαίων νοτών στην παρτιτούρα.

6. Υλοποίηση

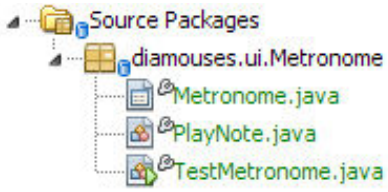
Στο κεφάλαιο αυτό αναλύονται τα κομμάτια του κώδικα που υλοποιήθηκαν στα πλαίσια της πτυχιακής εργασίας, καθώς και οι τροποποιήσεις που έγιναν στον αρχικό κώδικα. Επίσης, παρουσιάζονται εργαλεία ανοιχτού κώδικα που χρησιμοποιήθηκαν όπως το Apache Ant, για τη διαδικασία packaging την εφαρμογής, και το Joomla, για την δημιουργία της ιστοσελίδας και την ενσωμάτωση της εφαρμογής σε αυτή.

6.1 Μετρονόμος

Για την υλοποίηση του μετρονόμου θα χρησιμοποιηθεί κώδικας από το project JFrets [11]. Τα απαραίτητα αρχεία για την εμφάνιση του παραθύρου του Μετρονόμου είναι τα `Metronome` και `PlayNote`.

Αφού εντοπίστηκε ο κώδικας των δύο αυτών αρχείων και αντιγράφηκε στο project της πτυχιακής εργασίας, δημιουργήθηκε ένα package ώστε να είναι τακτοποιημένα τα αρχεία της εφαρμογής. Το νέο πακέτο ονομάστηκε `diamouses.ui.Metronome` και θα περιέχει όλα τα αρχεία τα σχετικά με το Μετρονόμο.

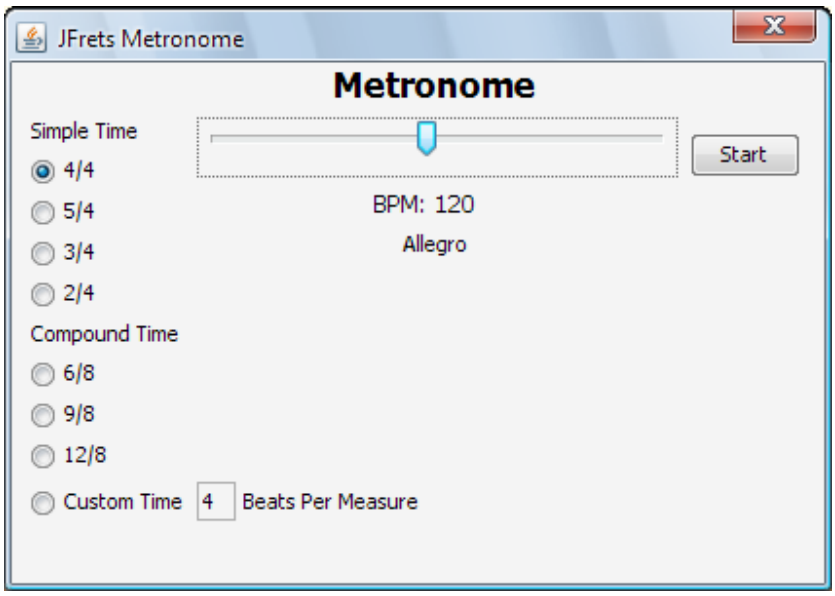
Επίσης υλοποιήθηκε ένα νέο αρχείο στο ίδιο πακέτο με το όνομα `TestMetronome` το οποίο αρχικοποιεί το μετρονόμο για να γίνουν δοκιμές. Έτσι το τελικό αποτέλεσμα – package structure καθώς και ο κώδικας του `TestMetronome.java` παρουσιάζονται στο Σχήμα 6-1.



Σχήμα 6-1: Package structure μετρονόμου.

```
package diamouses.ui.Metronome;

import javax.swing.JFrame;

public class TestMetronome {

	public static void main (String [] args) {
		Metronome met = new Metronome(new JFrame(), null, true);
		met.setVisible(true);
	}
}
```

Στην παρακάτω εικόνα ο χρήστης επιλέγει το κουμπί Start και κατόπιν μπορεί να κάνει τις απαραίτητες ρυθμίσεις όσον αφορά στο τέμπο του μετρονόμου και στην ταχύτητά του (Beats Per Minute) όπως φαίνεται στο Σχήμα 6-2.



Σχήμα 6-2: Παράθυρο μετρονόμου.

6.2 Βελτίωση αρχείου: ScoreEditor.java

Η κύρια δουλειά του αρχείου ScoreEditor.java είναι να εμφανίζει menu με κουμπιά που απεικονίζεται στο Σχήμα 6-3, για επιλογή νότας ή παύσης καθώς και μουσικού οργάνου. Επίσης διαθέτει controls για επιλογή παρτιτούρας, για διαγραφή της τελευταίας νότας στην παρτιτούρα και για καθαρισμό όλης της παρτιτούρας.

Σχήμα 6-3: Μενού με κουμπιά του score editor.

Η επιλογή νότας και παύσης δεν λειτουργούσε σωστά. Ήταν υλοποιημένες σαν JToggleButtons αλλά ο χρήστης μπορούσε να κάνει πολλαπλές επιλογές και η δεύτερη νότα στο duration ήταν μόνιμα επιλεγμένη.

Η διαγραφή της τελευταίας νότας κράσαρε την εφαρμογή, όταν η παρτιτούρα ήταν άδεια όπως φαίνεται στο Σχήμα 6-4.

```
Exception in thread "AWT-EventQueue-0" java.lang.ArrayIndexOutOfBoundsException:
        at java.util.Vector.removeElementAt(Unknown Source)
        at jm.music.data.Phrase.removeLastNote(Phrase.java:584)
```

Σχήμα 6-4: Exception διαγραφής τελευταίας νότας σε άδεια παρτιτούρα.

Το αρχείο ScoreEditor.java περιείχε παραπάνω από 600 γραμμές κώδικα με δεκάδες μεταβλητές και μεθόδους, οι οποίες δεν ήταν ονοματισμένες έτσι ώστε να αποκαλύπτουν τη χρήση τους. Η βελτίωση του αρχείου αυτού, τόσο σχεδιαστικά όσο και λειτουργικά κρίθηκε απαραίτητη.

Βελτίωση 1

Ας δούμε τις δεκάδες μεθόδους jButton_ActionPerformed. Στην ισχύουσα υλοποίηση για κάθε κουμπί του παραπάνω μενού υπήρχε ο παρακάτω κώδικας:

```
jButton8.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton8ActionPerformed(evt);
    }
});
```

Και κατόπιν η αντίστοιχη μέθοδος:

```
// ##########################################
// ############### 1/16 note ################
// ##########################################
private void jToggleButton1ActionPerformed(java.awt.event.ActionEvent evt) {// GEN
                                                                            // -
                                                                            // FIRST
                                                                            // :
                                                                            // event_jToggleBut
    stave.setNoteDuration(0.25);
}// GEN-LAST:event_jToggleButton1ActionPerformed
```

Αντί να έχουμε δεκάδες μεθόδους, μπορούμε να χρησιμοποιήσουμε έναν άλλο τρόπο υλοποίησης. Σε κάθε κουμπί προσθέτουμε σαν ActionListener την ίδια την κλάση και θέτουμε ένα ActionCommand, ένα string βάση του οποίου θα αναγνωρίσουμε στη συνέχεια ποιο κουμπί πατήθηκε.

```
jButton8.setActionCommand("JBUTTON8");
jButton8.addActionListener(this);
```

Εφόσον για κάθε κουμπί αντικαταστήσαμε την πρώτη από την παραπάνω μέθοδο με τις παραπάνω δύο γραμμές και τη δεύτερη μέθοδο με δική μας υλοποίηση που φαίνεται παρακάτω, μειώθηκε κατά 150 γραμμές το αρχείο μας και οι απαραίτητες μέθοδοι κατά 8.

Βελτίωση 2

Παρατηρώντας τον κώδικα τώρα που άρχισε να είναι readable, αρκετός κώδικας επαναλαμβανόταν σε αρκετά σημεία του αρχείου. Για παράδειγμα στα rests 1/16 , 1/8 κτλ 12 γραμμές κώδικα επαναλαμβάνονται 6 φορές, μια για κάθε rest. Η μόνη διαφορά είναι μια τιμή τύπου double.

```
} else if (action_command.equals("JBUTTON4")) {
    // 1 / 16 rest
    Phrase phr = stave.getPhrase();
    Note n;
    if (stave.getSelectedNote() == -1) {
        n = new Note(-2147483648, 0.25);
        phr.add(n);
    } else {
        n = phr.getNote(stave.getSelectedNote());
        n.setPitch(-2147483648);
        n.setRhythmValue(0.25);
    }
    stave.repaint();
} else if (action_command.equals("JBUTTON5")) {
    // 1 / 8 rest
    Phrase phr = stave.getPhrase();
    Note n;
    if (stave.getSelectedNote() == -1) {
        n = new Note(-2147483648, 0.5);
        phr.add(n);
    } else {
        n = phr.getNote(stave.getSelectedNote());
        n.setPitch(-2147483648);
        n.setRhythmValue(0.5);
    }
    stave.repaint();
```

Ο παραπάνω κώδικας μπορούσε να βελτιωθεί με τη δημιουργία μιας μεθόδου που παίρνει σαν παράμετρο την επίμαχη τιμή και τρέχει τις δώδεκα εντολές.

```
private void setRest(double rest_value) {
    Phrase phr = stave.getPhrase();
    Note n;
    if (stave.getSelectedNote() == -1) {
        n = new Note(-2147483648, rest_value);
        phr.add(n);
    } else {
        n = phr.getNote(stave.getSelectedNote());
        n.setPitch(-2147483648);
        n.setRhythmValue(rest_value);
    }
    stave.repaint();
}
```

Με τη δημιουργία της παραπάνω μεθόδου μειώθηκε ο κώδικας του αρχείου ακόμα περισσότερο (50 γραμμές λιγότερες).

Βελτίωση 3

Δόθηκαν ονόματα στις μεταβλητές και δημιουργήθηκαν μέθοδοι για κάθε λειτουργία. Κατόπιν το actionPerformed, η μέθοδος που είναι υπεύθυνη για κάθε control (κουμπί της εφαρμογής) έγινε τόσο απλή και ξεκάθαρη.

```
public void actionPerformed(ActionEvent e) {
    // Get the action-command associated with the event
    String action_command = e.getActionCommand();

    if (action_command.equals("Note_16th"))
        stave.setNoteDuration(0.25);
    else if (action_command.equals("Note_8th"))
        stave.setNoteDuration(0.5);
    else if (action_command.equals("Note_4th"))
        stave.setNoteDuration(1.0);
    else if (action_command.equals("Note_Half"))
        stave.setNoteDuration(2.0);
    else if (action_command.equals("Note_Whole"))
        stave.setNoteDuration(4.0);
    else if (action_command.equals("Rest_16th"))
        setRest(0.25);
    else if (action_command.equals("Rest_8th"))
        setRest(0.5);
    else if (action_command.equals("Rest_4th"))
        setRest(1.0);
    else if (action_command.equals("Rest_Half"))
        setRest(2.0);
    else if (action_command.equals("Rest_Whole"))
        setRest(4.0);
    else if (action_command.equals("Select_Instrument"))
        selectProperInstrument();
    else if (action_command.equals("Clear_Stave"))
        stave.setPhrase(new Phrase());
    else if (action_command.equals("Remove_Last_Note"))
        removeLastNote();
    else if (action_command.equals("Show_Instrument_Dialog"))
        showInstrumentDialog();
    else if (action_command.equals("Play"))
        playTune();
}
```

Βελτίωση 4

Όταν ο χρήστης προσπαθούσε να σβήσει την τελευταία νότα μιας παρτιτούρας ενώ η παρτιτούρα ήταν άδεια, η εφαρμογή κόλλαγε. Μια μικρή αλλαγή στον κώδικα (που πλέον τοποθετήθηκε σε μια μέθοδο) ήταν αρκετή για να διορθωθεί το λάθος αυτό.

```
/**
 * Method that removes last note in the stave
 */
private void removeLastNote() {
    Phrase phrase = stave.getPhrase();
    if (phrase.getSize()>0) {
        phrase.removeLastNote();
        stave.setPhrase(phrase);
    }
}
```

Το if (phrase.getSize() > 0) ήταν αρκετό για να διορθωθεί το παραπάνω λάθος.

Βελτίωση 5

Δημιουργήθηκε μέθοδος για να δημιουργεί τα κουμπιά της εφαρμογής. Ξανά κώδικας επαναλαμβανόταν για κάθε κουμπί του ScoreEditor.

```
        /**
         * Method creates a "smart" JToggleButton
         * @param imageName The location of the image of this toggle-button
         * @param selected_imageName The location of the image of this toggle-button
         *                           when the toggle button is selected
         * @param actionCommand The action-command to set for this components
         * @param toolTipText the tooltip of the component
         * @param altText Alternative text in case the image is missing
         * @return A JToggleButton component
         */
        protected JToggleButton makeToggleButton(String imageName, String
selected_imageName, String actionCommand, String toolTipText, String altText) {
                // Look for the image.
                String imgLocation = "images/menu_images/" + imageName + ".gif";
                String imgLocation2 = "images/menu_images/" + selected_imageName + ".gif";

                URL imageURL = Main.class.getResource(imgLocation);
                URL imageURL2 = Main.class.getResource(imgLocation2);

                // Create and initialize the button.
                JToggleButton button = new JToggleButton();
                button.setActionCommand(actionCommand);
                button.setToolTipText(toolTipText);
                button.addActionListener(this);

                if ( (imageURL != null) & (imageURL2 != null) ){ // image found
                        button.setIcon(new ImageIcon(imageURL, altText));
                        button.setSelectedIcon(new ImageIcon(imageURL2, altText));
                } else { // no image found
                        button.setText(altText);
                        System.err.println("Resource not found: " + imgLocation + "" +
(imageURL==null) + " " + (imageURL2==null));
                }
                return button;
        }
```

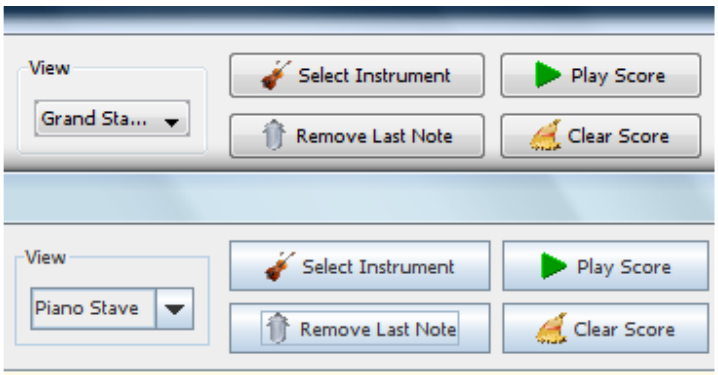
Γενικά το αρχείο ScoreEditor.java είχε παραπάνω από 600 γραμμές κώδικα. Μετά από σειρά αλλαγών και βελτιώσεων κατέληξε να κάνει ακριβώς την ίδια δουλειά πολύ πιο αποτελεσματικά τόσο από πλευράς χρήσης μνήμης όσο και από λειτουργικότητα με λιγότερο από 300 γραμμές κώδικα.

6.3 Βελτίωση αρχείου: Main.java

Μια αδυναμία που εντοπίζεται αμέσως στο αρχείο αυτό, είναι ότι το Look & Feel είναι το default της Java. Γνωρίζοντας ότι η Java παρέχει δυνατότητες να προσομοιάσει το γραφικό περιβάλλον πολλών λειτουργικών συστημάτων μπορούμε να βελτιώσουμε αισθητά το Look & Feel της εφαρμογής προσθέτοντας την παρακάτω μέθοδο, και καλώντας την στη main προτού εμφανίσουμε το γραφικό περιβάλλον.

```
/**
* setDefault() , Sets the default Look and Feel.
*  On Microsoft Windows platforms, this specifies the Windows Look & Feel.
* On Mac OS platforms, this
*  specifies the Mac OS Look & Feel. On Sun platforms, it specifies the CDE/Motif
* Look & Feel.
*
*  With JDK1.4.2 upwards it sets the 'Luna' look and feel in Windows XP and
* 'Bluecurve' in GNOME.
*/
 public static void setDefault() {
     try {
         UIManager.setLookAndFeel( UIManager.getSystemLookAndFeelClassName() );
     } catch (java.lang.ClassNotFoundException classE) {
         // Handle exception
     } catch (java.lang.InstantiationException insE) {
         //Handle exception
     } catch (java.lang.IllegalAccessException illE) {
         //Handle exception
     } catch (javax.swing.UnsupportedLookAndFeelException unE) {
         //Handle exception
     }
 }
```

Η παραπάνω μέθοδος «κοιτάει» σε ποιο λειτουργικό σύστημα τρέχει η εφαρμογή και θέτει το κατάλληλο Look & Feel.

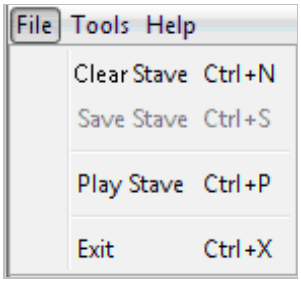


Σχήμα 6-5: Look & Feel εφαρμογής.

Στο Σχήμα 6-5 παρουσιάζεται στο πάνω μέρος το νέο Look & Feel που μοιάζει περισσότερο σε αυτό που χρησιμοποιούν τα Windows XP/Vista από το αρχικό το οποίο φαίνεται από κάτω σαν μέτρο σύγκρισης.

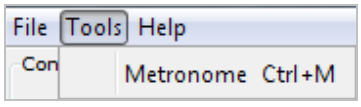
Στο αρχείο αυτό προστέθηκαν επίσης αρκετές γραμμές κώδικα (περίπου 150), με σκοπό τη δημιουργία μενού επιλογών.

Στο μενού File υπάρχουν επιλογές για 'Καθαρισμό Παρτιτούρας' και 'Παίξιμο παρτιτούρας' όπως φαίνεται στο Σχήμα 6-6.



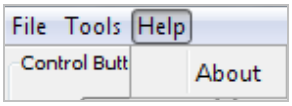
Σχήμα 6-6: Μενού File.

Στο Σχήμα 6-7 απεικονίζεται το μενού Tools όπου υπάρχει η επιλογή για τον 'Μετρονόμο'.



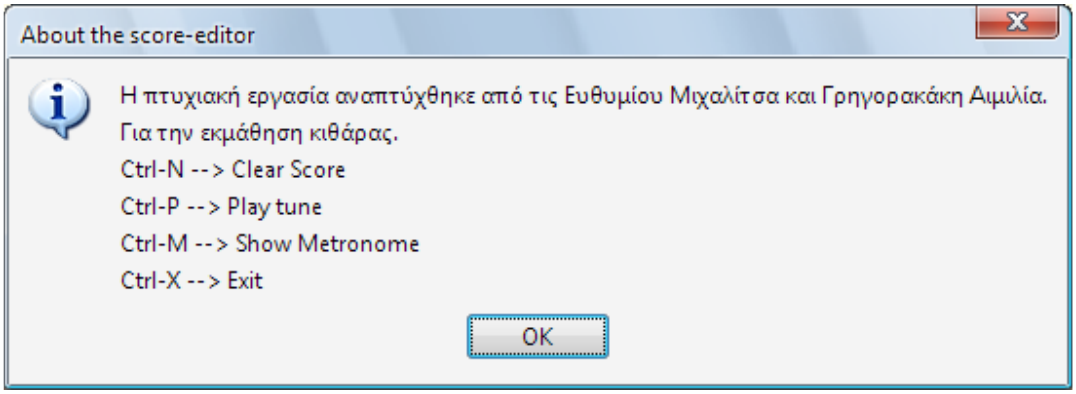
Σχήμα 6-7: Μενού Tools.

Ενώ στο μενού Help υπάρχει η επιλογή 'About'(Σχήμα 6-8).



Σχήμα 6-8: Μενού Help.

Όταν ο χρήστης επιλέξει το Help → About εμφανίζεται στο χρήστη το μήνυμα που παρουσιάζεται στο Σχήμα 6-9.



Σχήμα 6-9: Μήνυμα στο 'About'.

Τα short-cuts που εμφανίζονται στο About Dialog λειτουργούν κανονικά στην εφαρμογή καθώς υλοποιήθηκαν στο Main.java

Ο κώδικας του Main.java αρχικοποιεί την εφαρμογή. Είναι τύπου JFrame και στη main μέθοδο, την μέθοδο που καλείται όταν τρέχει η κλάση αυτή καλείται ο constructor της Main, ο οποίος θέτει τίτλο στο JFrame αρχικοποιεί τις μεταβλητές και κατόπιν τοποθετεί τα τρία κομμάτια που αποτελούν την εργασία αυτή (το πάνελ επιλογών, το score που είναι η παρτιτούρα και την εικονική κιθάρα) στην οθόνη του χρήστη.

```
	/**
	 * Constructor
	 */
	public Main() {
		this.setTitle("Πτυχιακή ΤΕΙ Κρήτης -  Ευθυμίου Μιχαλίτσα,
Γρηγορακάκη Αιμιλία");
		initComponents();
		score = new ScoreEditor();
		setJMenuBar(getJMenuBar());

		Toolkit kit = getToolkit();
		Dimension screenSize = kit.getScreenSize();
		int screenWidth = screenSize.width;
		//-System.out.println("screenwidth" + screenWidth);

		GuitarNeck ff = GuitarNeck.getInstance(screenWidth);
		getContentPane().add(score, BorderLayout.CENTER);
		getContentPane().add(score.getJToolBar(),
BorderLayout.PAGE_START);
		getContentPane().add(ff, BorderLayout.SOUTH);
		setVisible(true);
		setExtendedState(JFrame.MAXIMIZED_HORIZ);
		pack();
	}
```

Το πάνελ επιλογών παρουσιάστηκε στο προηγούμενο κεφάλαιο 7.2. Η εικονική κιθάρα και η παρτιτούρα θα παρουσιαστούν σε επόμενα κεφάλαια. Η άλλη εργασία που κάνει η Main είναι η προσθήκη του Μενού Επιλογών.

Η μέθοδος που δημιουργεί το μενού επιλογών ακολουθεί:

```
	/**
	 * Creates a JMenuBar (File/Tools/Help) with appropriate
	 * menu items
	 */
	public JMenuBar getJMenuBar() {
		// Local Variables
		JMenuBar menuBar;
		JMenu menu;
		JMenuItem menuItem;

		// Create the menu bar.
		menuBar = new JMenuBar();
```

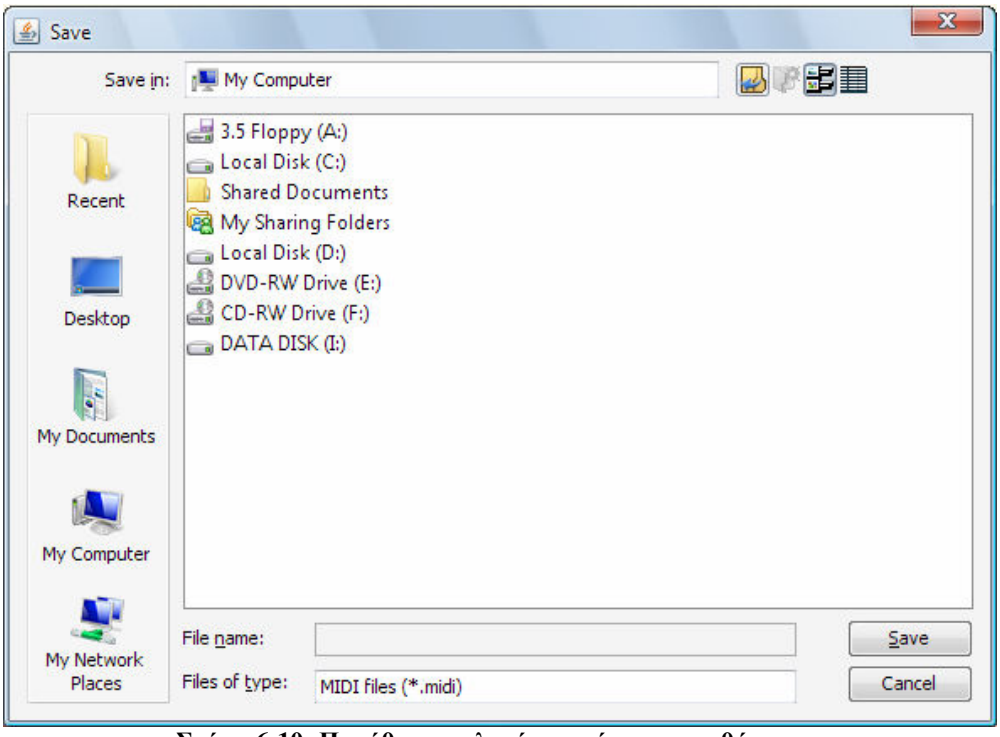
```
            // 1. Build the 'File' menu.
            menu = new JMenu("File");
            menu.setMnemonic(KeyEvent.VK_F);
            menuBar.add(menu);

            // 'Clear Stave' MenuItem
            menuItem = new JMenuItem("Clear Stave", KeyEvent.VK_N);

      menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,Action
Event.CTRL_MASK));
            menuItem.setActionCommand("Clear_Stave");
            menuItem.addActionListener(this);
            menu.add(menuItem);
```

Κάθε μενού αποτελείται από menu-items. Τα menu-items έχουν μια ονομασία, «ακούν» σε ένα KeyEvent και έχουν κάποιο Accelarator. Με τη χρήση των accelerators ο χρήστης μπορεί να πατήσει ένα συνδυασμό πλήκτρων στο πληκτρολόγιό του π.χ. Ctrl-P για να κάνει Play το tune της παρτιτούρας.

Το κάθε menu-item κάνει register στον ActionListener ο οποίος περιέχει τον απαραίτητο κώδικα για την υλοποίηση της επιλογής του χρήστη. Για παράδειγμα για την αποθήκευση της παρτιτούρας ως MIDI αρχείο στο δίσκο απαιτείται ο παρακάτω κώδικας.

```
if (action_command.equals("Save_Stave")) {
   JFileChooser fc = new JFileChooser(new File(filename));
 // Show save dialog; this method does not return until the dialog is
closed
   fc.showSaveDialog(this);
   File selected_file = fc.getSelectedFile();

System.out.println("Going to save score into file : " +
selected_file.toString());

      Score s = new Score(new Part(score.stave.getPhrase()));
      Write.midi(s, selected_file.toString());
}
```

Κώδικας από [30] και [31]. Εμφανίζει στο χρήστη το παράθυρο επιλογής αρχείου(βλέπε Σχήμα 6-10) για αποθήκευση και κατόπιν αποθηκεύει σε αυτό το αρχείο το MIDI που αντιστοιχεί στην παρτιτούρα.



Σχήμα 6-10: Παράθυρο επιλογής αρχείου για αποθήκευση.

Μετά την επιλογή του φακέλου αλλά και του ονόματος του αρχείου που έχει την κατάληξη .midi εμφανίζεται στην κονσόλα το μήνυμα:

```
Going to save score into file : C:\ test2.midi
---------------------- Writing MIDI File -----------------------------
Converting to SMF data structure...
    Part 0 'Untitled Part' to SMF Track on Ch. 0:  Phrase 0:......
MIDI file 'C:\test2.midi' written from score 'Untitled Score' in 0.074
seconds.
----------------------------------------------------------------------
```

Και ένα valid αρχείο με κατάληξη .midi δημιουργείται και μπορούμε να το ακούσουμε με οποιαδήποτε media player (Winamp, Windows Media Player, VLC, Real Audio κτλ)

6.4 Αρχείο: Constants.java

Το αρχείο Constants.java (50 γραμμές) υλοποιήθηκε ως βοηθητικό αρχείο για τη διευκόλυνση του loading αρχείων εικόνας. Στο Σχήμα 6-11 παρουσιάζονται οι εικόνες που αποφασίστηκε να χρησιμοποιηθούν στη συνέχεια της εργασίας για να εμφανίζονται στο μπράτσο της κιθάρας στην αντίστοιχη θέση. Δηλαδή όταν ο χρήστης επιλέγει την 3η θέση της 2ης χορδής της κιθάρας, η αντίστοιχη νότα θα «ζωγραφίζεται» στο αντίστοιχο σημείο της κιθάρας. Για προγραμματιστική διευκόλυνση όλα τα paths των εικόνων, αλλά και ο κώδικας για να τις φορτώνει τοποθετήθηκε στο αρχείο Constants.java

Σχήμα 6-11: Οι εικόνες του Constants.java.

```
package diamouses.ui.scoreEditor;

import javax.swing.ImageIcon;

public final class Constants
{
    public  ImageIcon a;
    public  ImageIcon asharp;
    public  ImageIcon b ;
    public  ImageIcon c;
    public  ImageIcon csharp;
    public  ImageIcon d;
    public  ImageIcon dsharp;
    public  ImageIcon e;
    public  ImageIcon f;
    public  ImageIcon fsharp;
    public  ImageIcon g;
    public  ImageIcon gsharp;

    public Constants()
    {
        a = new ImageIcon(getClass().getResource("images/note_images/a.png"));
        asharp = new ImageIcon(getClass().getResource("images/note_images/asharp.png"));
        b = new ImageIcon(getClass().getResource("images/note_images/b.png"));
        c = new ImageIcon(getClass().getResource("images/note_images/c.png"));
        csharp = new ImageIcon(getClass().getResource("images/note_images/csharp.png"));
        d = new ImageIcon(getClass().getResource("images/note_images/d.png"));
        dsharp = new ImageIcon(getClass().getResource("images/note_images/dsharp.png"));
        e = new ImageIcon(getClass().getResource("images/note_images/e.png"));
        f = new ImageIcon(getClass().getResource("images/note_images/f.png"));
        fsharp = new ImageIcon(getClass().getResource("images/note_images/fsharp.png"));
        g = new ImageIcon(getClass().getResource("images/note_images/g.png"));
        gsharp = new ImageIcon(getClass().getResource("images/note_images/gsharp.png"));
        e.setDescription("e");
        f.setDescription("f");
        fsharp.setDescription("fsharp");
```

```
        g.setDescription("g");
        gsharp.setDescription("gsharp");
        a.setDescription("a");
        asharp.setDescription("asharp");
        b.setDescription("b");
        c.setDescription("c");
        csharp.setDescription("csharp");
        d.setDescription("d");
        dsharp.setDescription("dsharp");

    }

}
```

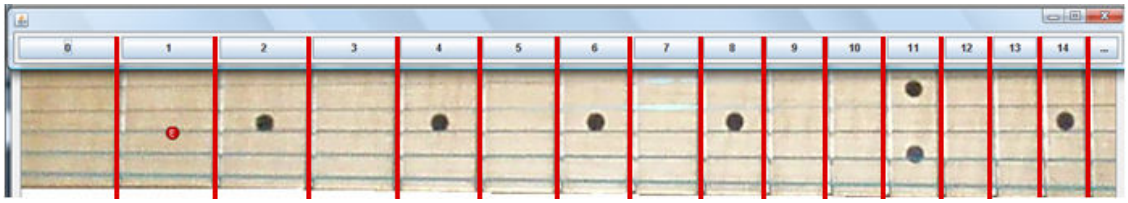
6.5 Αρχείο: GuitarNeck.java

Ένας από τους αρχικούς στόχους της πτυχιακής αυτής εργασίας ήταν η εμφάνιση μέρους της κιθάρας και συγκεκριμένα του μπράτσου της και η υλοποίηση κατάλληλης διεπαφής για την αλληλεπίδραση του χρήστη με την εικονική απεικόνιση της κιθάρας. Επίσης ένας άλλος στόχος της εφαρμογής της πτυχιακής ήταν η αλληλεπίδραση μεταξύ της παρτιτούρας και της εικονικής κιθάρας.

Στο Σχήμα 6-12 εμφανίζεται η εικονική κιθάρα όπως έχει υλοποιηθεί από το project JFrets. Ακριβώς πάνω από την κιθάρα αυτή εμφανίζεται το πρώτο prototype της δικής μας υλοποίησης με έναν αριθμό κουμπιών (Java JButtons) των οποίων το πλάτος έχει ορισθεί ίσο με το πλάτος του κάθε τάστου της κιθάρας.

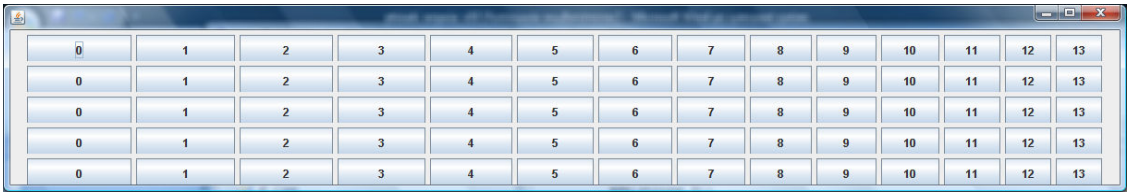
Σχήμα 6-12: Εικονική κιθάρα JFrets.

Για επιβεβαίωση των μεγεθών των κουμπιών που υλοποιήθηκαν, κόκκινες κάθετες γραμμές εμφανίζονται κατόπιν επεξεργασίας της εικόνας με πρόγραμμα επεξεργασίας (βλέπε Σχήμα 6-13).



Σχήμα 6-13: Απεικόνιση τωνμεγεθών των κουμπιών που υλοποιήθηκαν.

Επαναλαμβάνοντας το JPanel του πρώτου prototype 5 φορές (5 * 13 κουμπιά) εμφανίζεται το αποτέλεσμα του Σχήματος 6-14 που μοιάζει να πλησιάζει την επιθυμητή μορφή. Δηλαδή τα κουμπιά αποτελούν τους χώρους που επιθυμούμε να αλληλεπιδρά ο χρήστης. Στόχος είναι ο χρήστης να βλέπει μόνο το μπράτσο της κιθάρας και κάνοντας mouse-click πάνω σε κάποιο σημείο της να γνωρίζει η εφαρμογή ποια ακριβώς νότα πατήθηκε.



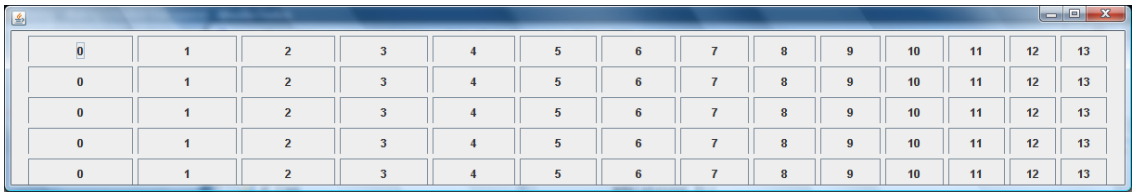
Σχήμα 6-14: Πρώτο prototype της εικονικής κιθάρας.

Τα κουμπιά ουσιαστικά θα βρίσκονται «πίσω» από την εικόνα της κιθάρας και ο τελικός χρήστης θα αγνοεί την ύπαρξη τους. Θα είναι όμως ο μηχανισμός με τον οποίο θα καταλαβαίνει η εφαρμογή τα mouse-clicks.

Τα JButtons σαν components της swing library εμφανίζονται στην οθόνη του χρήστη επικαλύπτοντας το υπάρχον JPanel. Με την παρακάτω εντολή ορίζουμε σε κάθε κουμπί να μην «γεμίζει» το content-area. Δηλαδή να μην κάνει paint την επιφάνεια του κουμπιού που είναι γύρω από τις εικόνες ή το κείμενο του κουμπιού. Συνήθως αυτό γίνεται σε χρώμα συμβατό με αυτό του λειτουργικού συστήματος που τρέχει η εφαρμογή.

```
button.setContentAreaFilled(false);
```

Το οπτικό αποτέλεσμα απεικονίζεται στο Σχήμα 6-15.

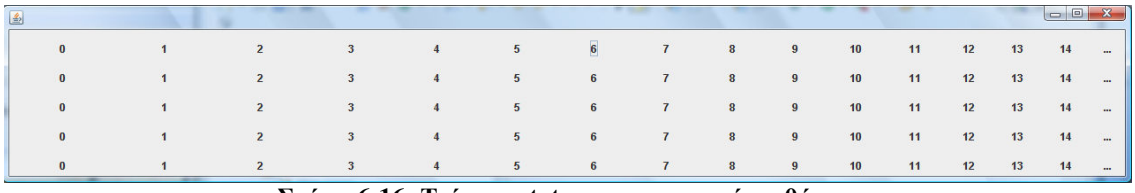


Σχήμα 6-15: Δεύτερο prototype της εικονικής κιθάρας.

Για να μην εμφανίζεται το διακοσμητικό border γύρω από τα κουμπιά η παρακάτω εντολή είναι απαραίτητη.

```
b1.setBorderPainted(false);
```

Το οπτικό αποτέλεσμα φαίνεται στο Σχήμα 6-16.



Σχήμα 6-16: Τρίτο prototype της εικονικής κιθάρας.

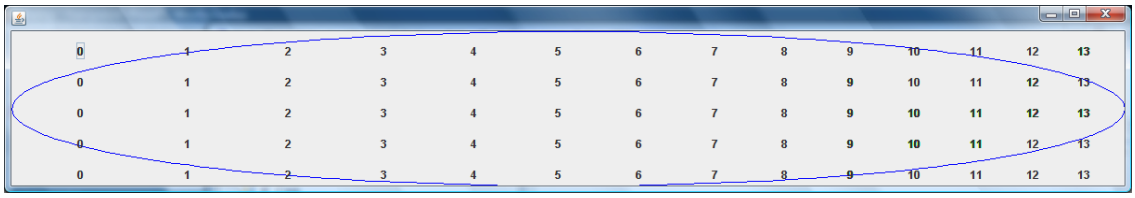
Θέλοντας να δοκιμάσουμε να εμφανίσουμε μια εικόνα πίσω από τα κουμπιά της παραπάνω εικόνας υλοποιήθηκε η παρακάτω μέθοδος που πρέπει να εμφανίσει ένα μπλε οβάλ σχήμα στο background.

```
public void paintComponent(Graphics g) {
      int width = getWidth();
      int height = getHeight();
      g.setColor(Color.blue);
      g.drawOval(0, 0, width, height);
}
```

Η μπλε γραμμή σε οβάλ σχήμα δεν εμφανίζεται. Κάνοντας όμως .setOpaque(false) όλα τα JPanels που περιέχουν τα κουμπιά:

```
string_1.setOpaque(false);
string_2.setOpaque(false);
string_3.setOpaque(false);
string_4.setOpaque(false);
string_5.setOpaque(false);
```

Στο Σχήμα 6-17 βλέπουμε ότι το background πλέον εμφανίζεται και όλα τα components που χρησιμοποιήσαμε είναι TRANSPARENT (διαφανή).

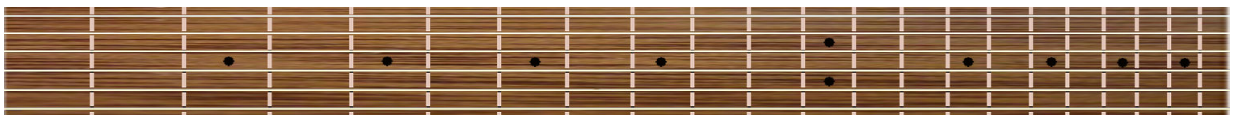


Σχήμα 6-17: Απεικόνιση οβάλ σχήματος στο prototype της εικονικής κιθάρας.

Διαθέτοντας την απαραίτητη τεχνογνωσία, ήρθε η ώρα να κάνουμε την πλήρη υλοποίηση της εικονικής κιθάρας. Ξεκινώντας από το wood texture που απεικονίζεται στο Σχήμα 6-18.

Σχήμα 6-18: Wood texture ταστίερας της κιθάρας.

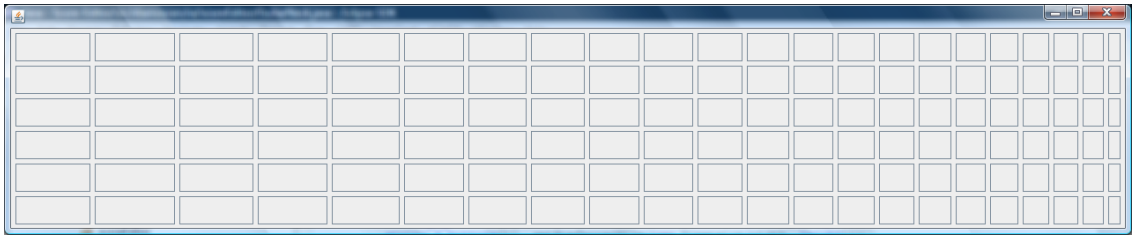
Και με ορισμένη δουλειά σε πρόγραμμα επεξεργασίας εικόνας καταλήγουμε στο Σχήμα 6-19 που απεικονίζει ένα ξύλινο μπράτσο κιθάρας με 6 χορδές και τα ανάλογα τάστα.



Σχήμα 6-19: Εικονική αναπαράσταση μπράτσου κιθάρας.

Η παραπάνω εικόνα θα χρησιμοποιηθεί σαν background για την υλοποίηση της εικονικής κιθάρας.

Δημιουργώντας ξανά τα απαραίτητα κουμπιά της εφαρμογής έχουμε το αποτέλεσμα που φαίνεται στο Σχήμα 6-20.



Σχήμα 6-20: Πάνελ κουμπιών εικονικής κιθάρας.

Εμφανίζουμε το background image με τον παρακάτω κώδικα πίσω από τα κουμπιά αυτά:

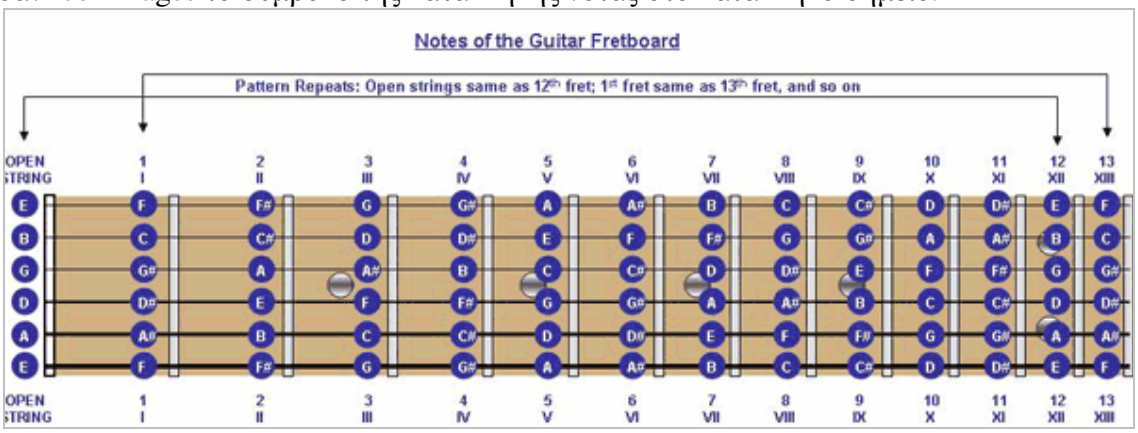
```
/**
 * Method Paint the Guitar Neck on the Background of this JPanel.
 */
public void paintComponent(Graphics g) {
    String imgLocation2 = "images/GuitarNeck.png";
    URL imageURL = GuitarNeck.class.getResource(imgLocation2);
    ImageIcon ii = new ImageIcon(imageURL, "altText");
    g.drawImage(ii.getImage(),
0,0,this.getSize().width,this.getSize().height, this);
}
```

έχουμε το οπτικό αποτέλεσμα που παρουσιάζεται στο Σχήμα 6-21.

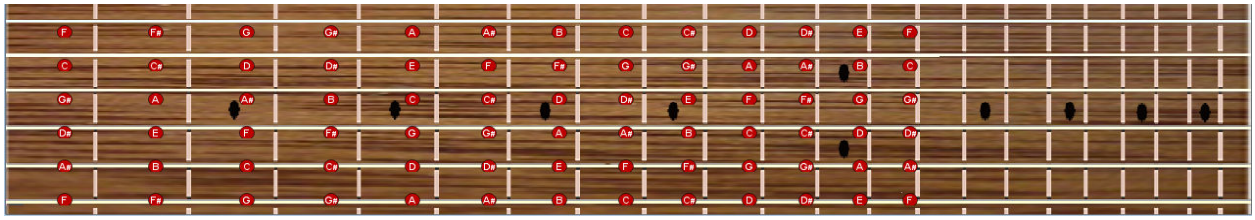
Σχήμα 6-21: Πάνελ εικονικής κιθάρας.

Μπροστά από την κάθε θέση της κιθάρας «κρύβεται» ένα διαφανές JButton.

Έχοντας κατά νου τις θέσεις της κιθάρας και τις νότες που αντιστοιχούν (βλέπε Σχήμα 6-22), και κάνοντας χρήση της κλάσης Constants.java που παρουσιάστηκε νωρίτερα θέτουμε σαν IconImages το σύμβολο της κατάλληλης νότας στο κατάλληλο σημείο.



Σχήμα 6-22: Νότες στην ταστιέρα της κιθάρας.



Σχήμα 6-23: Διεπαφή εικονικής κιθάρας.

Υλοποιώντας τα παραπάνω σαν JToggleButtons αντί για απλά JButtons και θέτοντας την κατάλληλη εικόνα για τα «πατημένα» κουμπιά και καμία εικόνα για τα «μη-πατημένα» κουμπιά, έχουμε μια πρώτη διεπαφή χρήστη (βλέπε Σχήμα 6-23).

Η υλοποίηση του κώδικα ακολούθησε τις αρχές του αντικειμενοστραφούς προγραμματισμού. Αντί να επαναλαμβάνεται κώδικας με πλήθος συντεταγμένων υλοποιήθηκαν ορισμένες μέθοδοι (4 μέθοδοι) που η κάθε μια κάνει συγκεκριμένη εργασία.

1. Η μέθοδος `getIconsForString(int string)` παίρνει σαν παράμετρο την χορδή της κιθάρας, τιμές από ένα (1) μέχρι και έξι (6) και επιστρέφει πίνακα αντικειμένων `javax.swing.ImageIcon` με τις εικόνες των νοτών.

```
/**
 * Method getIconsForString returns an Array [] of ImageIcons
 * with the images of the appropriate Notes for each string of
 * the Guitar.
 *
 * @param string An integer from 1 to 6 representing the First..Sixth string
 *        of a Guitar.
 * @return an ImageIcon [] array.
 */
private ImageIcon [] getIconsForString( int string ) {
        Constants c = new Constants();
        switch (string) {
        case 1: ImageIcon [] string_1_notes = {c.e, c.f, c.fsharp, c.g,
                c.gsharp,c.a,c.asharp,c.b, c.c, c.csharp,c.d, c.dsharp,c.e,c.f,
                null, null, null, null,null,null,null,null};
        return string_1_notes;
        case 2: ImageIcon [] string_2_notes = {c.b, c.c,c.csharp,c.d,
                 c.dsharp,c.e,c.f, c.fsharp, c.g, c.gsharp,
                 c.a,c.asharp,c.b, c.c,null,null, null, null,null,null,null,null};
        return string_2_notes;
        case 3: ImageIcon [] string_3_notes = {c.g, c.gsharp,c.a,c.asharp,c.b,
                c.c,c.csharp,c.d, c.dsharp, c.e,c.f,c.fsharp, c.g,
                c.gsharp,null,null, null, null,null,null,null,null};
        return string_3_notes;
        case 4: ImageIcon [] string_4_notes = {c.d, c.dsharp, c.e,c.f,c.fsharp,
                c.g, c.gsharp, c.a,c.asharp,c.b, c.c, c.csharp,c.d,
                c.dsharp,null,null, null, null,null,null,null,null};
        return string_4_notes;
        case 5: ImageIcon [] string_5_notes = {c.a,c.asharp,c.b, c.c,c.csharp,c.d,
                c.dsharp, c.e,c.f,c.fsharp, c.g, c.gsharp,c.a, c.asharp,null,null,
                null, null,null,null,null,null};
        return string_5_notes;
        case 6: ImageIcon [] string_6_notes = { c.e,c.f,c.fsharp, c.g,
                c.gsharp,c.a, c.asharp,c.b, c.c,c.csharp,c.d, c.dsharp,
                c.e,c.f,null,null, null, null,null,null,null,null};
        return string_6_notes;
        default: System.out.println("Errornous string " + string +
                " in method GuitarFrame.getIconsForString");
        return null;
        }
}
```

2. Η μέθοδος `paintComponent()` που «ζωγραφίζει» το background της εικονικής κιθάρας:

```
/**
 * Method Paint the Guitar Neck on the Background of this JPanel.
 */
public void paintComponent(Graphics g) {
   String imgLocation2 = "images/GuitarNeck.png";
   URL imageURL = GuitarNeck.class.getResource(imgLocation2);
   ImageIcon ii = new ImageIcon(imageURL, "altText");
   g.drawImage(ii.getImage(), 0,0,this.getSize().width,this.getSize().height,
this);
}
```

3. Η μέθοδος setSizes() που δίνει στα διάφανα κουμπιά το κατάλληλο μέγεθος ώστε να πιάνουν τα mouse-clicks του χρήστη με όσο το δυνατόν μεγαλύτερη πιστότητα.

```
/**
 * This method sets the sizes of the JToggleButtons
 */
private void setSizes () {
       int width;
       for (int i = 0; i <= 21; i++) {
       switch (i) {
              case 0: width = 83; break;
              case 1: width = 89; break;
              case 2: width = 82; break;
              case 3: width = 77; break;
              case 4: width = 75; break;
              case 5: width = 66; break;
              case 6: width = 64; break;
              case 7: width = 59; break;
              case 8: width = 55; break;
              case 9: width = 54; break;
              case 10: width = 49; break;
              case 11: width = 46; break;
              case 12: width = 43; break;
              case 13: width = 40; break;
              case 14: width = 38; break;
              case 15: width = 35; break;
              case 16: width = 32; break;
              case 17: width = 30; break;
              case 18: width = 28; break;
              case 19: width = 26; break;
              case 20: width = 22; break;
              default: width = 12; break;
       }
       width+=2;
       width = screen_width*width/1280;
       string_1_buttons[i] .setPreferredSize(new Dimension(width, height));
       string_2_buttons[i] .setPreferredSize(new Dimension(width, height));
       string_3_buttons[i] .setPreferredSize(new Dimension(width, height));
       string_4_buttons[i] .setPreferredSize(new Dimension(width, height));
       string_5_buttons[i] .setPreferredSize(new Dimension(width, height));
       string_6_buttons[i] .setPreferredSize(new Dimension(width, height));
       }
}
```

Μάλιστα στην παραπάνω μέθοδο πέρα από το γεγονός ότι τα διάφανα κουμπιά αποκτούν το κατάλληλο πλάτος σε pixels με ξεκάθαρο τρόπο έχει υλοποιηθεί δυνατότητα για να μπορεί η εφαρμογή να τρέχει άψογα σε όλες τις δυνατές αναλύσεις.

Με άλλα λόγια η εικόνα που χρησιμοποιήθηκε για την εφαρμογή μας έχει διαστάσεις 1517 x 144 εικονο-στοιχεία (pixels) έτσι ώστε η εφαρμογή να είναι άριστη οπτικά ακόμα και σε αναλύσεις 1600x1200 ή και μεγαλύτερες. Επειδή όμως η ανάπτυξη της εφαρμογής έγινε σε οθόνη με ανάλυση 1280x800 και το πλάτος των κουμπιών σε pixels μετρήθηκε στην ανάλυση αυτή, έπρεπε να προβλεφθεί χρήση σε διαφορετικές αναλύσεις. Έτσι με έξυπνο τρόπο υλοποιήθηκε και δοκιμάστηκε η εφαρμογή και σε άλλες αναλύσεις με επιτυχία.

4. Μέθοδος `private JPanel getStringPanel(JToggleButton [] string_buttons, int string).`

```
/**
 * Method returns a JPanel with JButtons for each note of the Guitar
 * @param string First..Sixth string of the guitar
 * @return a JPanel with appropriate JButtons
 */
private JPanel getStringPanel(JToggleButton [] string_buttons, int string) {
	JPanel main_panel = new JPanel();
	main_panel .setLayout(new FlowLayout());
	ImageIcon [] string_images = getIconsForString(string);

	for (int i = 0; i <= 21; i++) {
		string_buttons[i] = new JToggleButton();
		string_buttons[i].setContentAreaFilled(false);
		string_buttons[i].setBorderPainted(false);
		string_buttons[i].setOpaque(false);
		string_buttons[i].setIcon(new ImageIcon());
		string_buttons[i].setSelectedIcon(string_images[i]);
		main_panel.add(string_buttons[i]);
	}
	return main_panel;
}
```

Η παραπάνω μέθοδος παίρνει σαν παραμέτρους ένα uninitialized πίνακα από Buttons και τον αριθμό μιας χορδής. Περιέχει κώδικα για να αρχικοποιεί τα κουμπιά και καθορίζει τις παραμέτρους για να είναι διαφανή (transparent) τα κουμπιά αυτά.

Τοποθετώντας τον κώδικα της εφαρμογής σε μεθόδους έχουμε τη δυνατότητα εύκολα να μετατρέψουμε τα components από JButtons σε JToggleButtons ή JLabels για να πειραματιστούμε και να ανακαλύψουμε την ιδανικότερη υλοποίηση.

Αφού έχουν υλοποιηθεί όλες οι απαραίτητες μέθοδοι παρουσιάζουμε τις μεταβλητές της κλάσης και τον constructor που αρχικοποιεί το νέο αυτό αντικείμενο-component.

Οι μεταβλητές του αντικειμένου είναι:

```
/**
 * Global variables for the application
 */
```

```
private int height = 32;
private int screen_width;

private JToggleButton [] string_1_buttons = new JToggleButton [22];
private JToggleButton [] string_2_buttons = new JToggleButton [22];
private JToggleButton [] string_3_buttons = new JToggleButton [22];
private JToggleButton [] string_4_buttons = new JToggleButton [22];
private JToggleButton [] string_5_buttons = new JToggleButton [22];
private JToggleButton [] string_6_buttons = new JToggleButton [22];
```

Ο αριθμός των global μεταβλητών είναι μικρός και με τέτοιο τρόπο ώστε να παραμετροποιείται εύκολα το αντικείμενο. Το height είναι το ύψος των κουμπιών (ή των χορδών) σε pixels, το screen_width είναι το πλάτος της εφαρμογής σε pixes και οι έξης πίνακες με JToggleButtons περιέχουν τα διάφανα κουμπιά για κάθε μια από τις χορδές της κιθάρας.

Τέλος, ο constructor του αντικειμένου είναι ο εξής:

```
/**
 * Constructor.
 * Initializes 6 JPanels, one for each guitar string
 */
public GuitarNeck(int screen_width) {
	this.screen_width = screen_width;

	// 1. Create six JPanels
	JPanel string_1 = getStringPanel(string_1_buttons,1);
	JPanel string_2 = getStringPanel(string_2_buttons,2);
	JPanel string_3 = getStringPanel(string_3_buttons,3);
	JPanel string_4 = getStringPanel(string_4_buttons,4);
	JPanel string_5 = getStringPanel(string_5_buttons,5);
	JPanel string_6 = getStringPanel(string_6_buttons,6);

	// 2. Make them Not-Opaque (Make them TRANSPARENT)
	string_1.setOpaque(false);
	string_2.setOpaque(false);
	string_3.setOpaque(false);
	string_4.setOpaque(false);
	string_5.setOpaque(false);
	string_6.setOpaque(false);

	// 3. Add each of the JPanel in a stack from top to bottom
	setLayout(new GridLayout(0,1,0,0));
	add(string_1);
	add(string_2);
	add(string_3);
	add(string_4);
	add(string_5);
	add(string_6);

	setSizes();
}
```

Παίρνει σαν παράμετρο το μέγεθος του παραθύρου. Κατόπιν αρχικοποιεί έξι JPanels κάνοντας χρήση της μεθόδου getStringPanel και στη συνέχεια κάνει transparent (διάφανα) τα νέα αυτά JPanels. Τέλος χρησιμοποιεί τον GridLayout Layout Manager με παράμετρους 0,1,0,0 εννοώντας τοποθέτηση όλων των components σε 1 στήλη με border 0,0.

Κάνοντας add τα 6 JPanels, τοποθετούνται όλα τα JPanels της εφαρμογής το ένα κάτω από το άλλο και τέλος καλείται η μέθοδος setSizes() για να βελτιστοποιήσουμε την εμφάνιση της εφαρμογής σε όλες τις πιθανές αναλύσεις.

Ένα ακόμα σημείο που χρήζει αναφοράς είναι ο τρόπος με τον οποίο θα προστεθεί αυτό το Component μέσα σε ένα JFrame.

```
JFrame f = new JFrame();
f.setVisible(true);
f.setExtendedState(JFrame.MAXIMIZED_HORIZ);

Toolkit kit = f.getToolkit();
Dimension screenSize = kit.getScreenSize();
int screenWidth = screenSize.width;
System.out.println("screenwidth" + screenWidth);

GuitarNeck ff = new GuitarNeck(screenWidth);
f.getContentPane().add(ff);
f.pack();
```

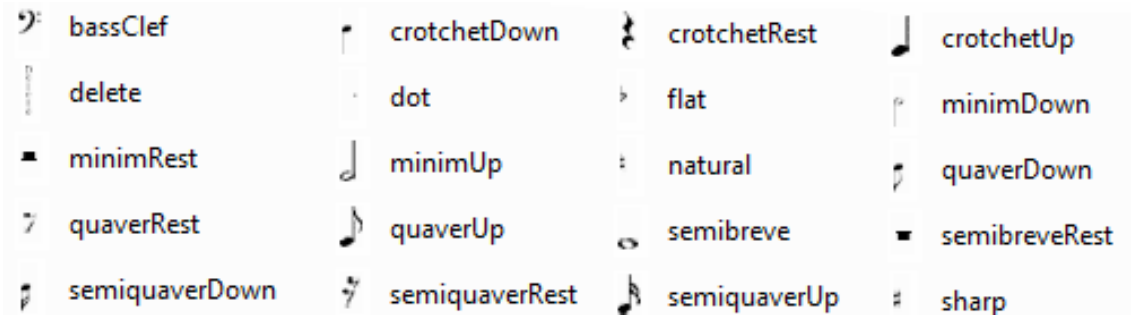
Αφού αρχικοποιήσουμε ένα νέο JFrame καλούμε την εντολή `setExtendedState(JFrame.MAXIMIZED_HORIZ);` ώστε να πάρει το μέγιστο δυνατό πλάτος το νέο παράθυρο. Κατόπιν με χρήση του εργαλείου Toolkit ανακαλύπτουμε το πλάτος της εφαρμογής (δηλαδή της ανάλυσης που θα τρέξει στο περιβάλλον του χρήστη) και αρχικοποιούμε το αντικείμενο GuitarNeck με την παράμετρο screenWidth.

Αυτό το σημείο αξίζει αναφοράς καθώς οι Layout Managers για να θέσουν σωστά τα μεγέθη (ύψος και πλάτος) σε κάθε component (και στο αντικείμενο αυτό έχουμε να θέσουμε τα μεγέθη ύψος και πλάτος σε 132 κουμπιά, 6 χορδές επί 22) πρέπει να το κάνουν τη στιγμή που τα αρχικοποιούμε για πρώτη φορά.

Για αυτό και είναι απαραίτητο να χρησιμοποιηθεί η παραπάνω τεχνική προτού αρχικοποιήσουμε το αντικείμενο μας.

6.6 Αρχείο: DStave.java

Η πτυχιακή αυτή εργασία απαιτούσε οι νότες στο πεντάγραμμο να αποκτούν κόκκινο χρώμα όταν ο χρήστης τις επέλεγε. Η πρώτη σκέψη ήταν να δημιουργούσαμε μια σειρά από εικόνες που αντί για μαύρο χρώμα να είχαν κόκκινο (βλέπε Σχήμα 6-24), να τις φορτώναμε στη μνήμη και να τις εμφανίζαμε όταν ο χρήστης κάνει click πάνω στην αντίστοιχη νότα.



Σχήμα 6-24: Μουσικά σύμβολα που έπρεπε να επαναδημιουργηθούν.

Η παραπάνω σκέψη απορρίφτηκε αμέσως. Οι λόγοι ήταν πολλοί:

1. Η εφαρμογή θα απαιτούσε πολύ παραπάνω αποθηκευτικό χώρο τόσο στη μνήμη όσο και στον αποθηκευτικό δίσκο.
2. Τι θα συνέβαινε αν κάποιος επέλεγε να τροποποιήσει τις αρχικές εικόνες. Θα έπρεπε να τροποποιήσει και την κόκκινη εκδοχή τους.
3. Τι θα συνέβαινε αν αντί για κόκκινο χρώμα θέλαμε να εμφανίζαμε τα επιλεγμένα μουσικά σύμβολα σε ΠΡΑΣΙΝΟ ή σε ΜΠΛΕ χρώμα; Θα έπρεπε να δημιουργηθούν νέες σειρές εικόνων, και θα απαιτούνταν περισσότερος κώδικας.

Η λύση που επιλέχθηκε και μετά από αρκετή έρευνα υλοποιήθηκε ήταν η μετατροπή του RGB color model δυναμικά όταν αυτό ήταν απαραίτητο. Για να γίνει αυτό, και καθώς η εικόνα που εμφανίζεται στην παρτιτούρα είναι τύπου java.awt.Image έπρεπε να δημιουργηθούν πρώτα δύο μέθοδοι. Η πρώτη θα μετέτρεπε το **java.awt.Image** σε **java.awt.Image.BufferedImage** και η άλλη θα έκανε την αντίστροφη διαδικασία.

Ο λόγος είναι ότι η κλάση java.awt.Image είναι μια πολύ απλή υλοποίηση που δεν περιέχει μεθόδους ούτε για Alpha Pixels (Transparency) ούτε για color models (RGB / ARGB κτλ).

Ας δούμε τη μέθοδο που μετατρέπει μια κλάση BufferedImage σε Image:

```
// This method returns an Image object from a buffered image
public static Image toImage(BufferedImage bufferedImage) {
  return Toolkit.getDefaultToolkit().createImage(bufferedImage.getSource());
}
```

Ας δούμε και τη μέθοδο που κάνει το αντίστροφο της παραπάνω, δηλαδή από Image σε BufferedImage:

```
// This method returns a buffered image with the contents of an image
public static BufferedImage toBufferedImage(Image image) {
   if (image instanceof BufferedImage) {
      return (BufferedImage)image;
   }

   // This code ensures that all the pixels in the image are loaded
   image = new ImageIcon(image).getImage();

   // Determine if the image has transparent pixels; for this method's
   // implementation, see e661 Determining If an Image Has Transparent Pixels
   // Create a buffered image with a format that's compatible with the screen
   BufferedImage bimage = null;
   GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
   try {
      // Determine the type of transparency of the new buffered image
      //  int transparency = Transparency.OPAQUE;
      //boolean hasAlpha = hasAlpha(image);
      //if (hasAlpha)
      int transparency = Transparency.BITMASK;

      // Create the buffered image
      GraphicsDevice gs = ge.getDefaultScreenDevice();
      GraphicsConfiguration gc = gs.getDefaultConfiguration();
      bimage = gc.createCompatibleImage(
      image.getWidth(null), image.getHeight(null), transparency);
    } catch (HeadlessException e) {
      // The system does not have a screen
    }

    if (bimage == null) {
      // Create a buffered image using the default color model
      //int type = BufferedImage.TYPE_INT_RGB;
      //if (hasAlpha)
      int type = BufferedImage.TYPE_INT_ARGB;
      bimage = new BufferedImage(image.getWidth(null), image.getHeight(null),
type);
    }

    // Copy image to buffered image
    Graphics g = bimage.createGraphics();

    // Paint the image onto the buffered image
    g.drawImage(image, 0, 0, null);
    g.dispose();

    return bimage;
}
```

Η παραπάνω μέθοδος ήταν πολύ πιο απαιτητική. Ο λόγος είναι ότι αντικείμενα τύπου `Image` δεν μπορούν να μετατραπούν σε αντικείμενα τύπου `BufferedImage`. Το πιο δυνατό ισοδύναμο είναι να δημιουργηθεί ένα νέο BufferedImage αντικείμενο και να «ζωγραφίσουμε» προγραμματιστικά την εικόνα του Image αντικειμένου πάνω σε αυτό, και αυτό ακριβώς υλοποιήσαμε παραπάνω.

Εφόσον οι δύο παραπάνω μέθοδοι τοποθετήθηκαν στο αρχείο DStave.java (και είναι μάλιστα στατικές μέθοδοι) μένει το κομμάτι που όταν μια νότα επιλέγεται να μετατρέπονται τα pixel της από μαύρο χρώμα σε κόκκινο (ή κάποιο άλλο).

Στη γραμμή 180 του αρχείου DGrantStave.java εντοπίζεται η εντολή

g.drawImage(currImage, totalBeatWidth, pitchTempPos, **this**);

Σε συνεργασία με τις εντολές

```
// draw notes and rests
for (int i = 0; i < phrase.size(); i++) {
   int notePitchNum = (int) phrase.getNote(i).getPitch();
   // choose graphic
   chooseImage(notePitchNum, phrase.getNote(i).getRhythmValue(), 71, 60, 50);
```

που προϋπάρχουν και για κάθε νότα της μουσικής φράσης Phrase παίρνουν την αντίστοιχη εικόνα με την εντολή **chooseImage** και την αποθηκεύουν στη μεταβλητή **currImage**, καταλαβαίνουμε ότι η εικόνα τύπου java.awt.Image περιέχει την εικόνα της κάθε νότας.

Έτσι πριν τη γραμμή 180 στο αρχείο DGrantStave.java τοποθετούμε το δικό μας κώδικα:

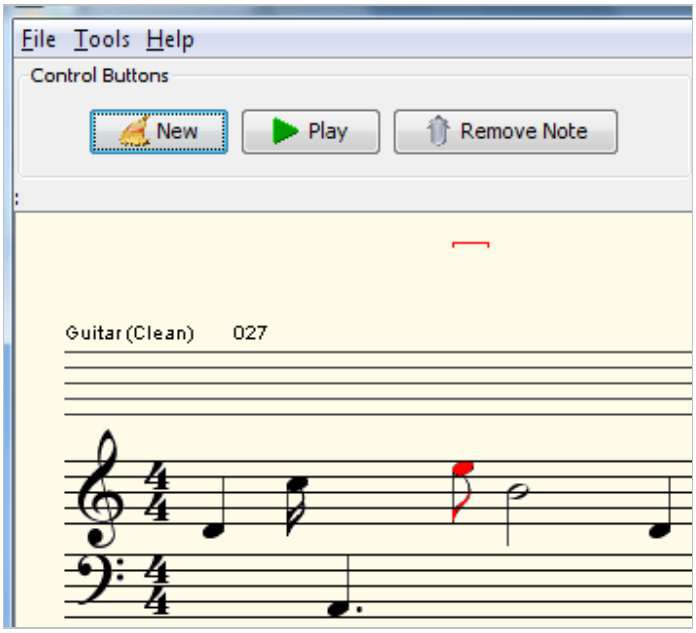
```
// Aimilia Grigorakaki - Euthimiou Michalitsa -> draw note/rest
if (currImage != null) {
   if (i==getSelectedNote()) {
      Note selected_note = phrase.getNote(i);//.getPitch();
      BufferedImage bi = DStave.toBufferedImage(currImage);
      int w = bi.getWidth();
      int h = bi.getHeight();
      int pixel;
      BufferedImage biOut = new BufferedImage(w, h,
      bi.TYPE_INT_ARGB_PRE);

      for (int x = 0; x < w; x++) {
         for (int y = 0; y < h; y++) {
            pixel = bi.getRGB(x, y);
            if (pixel == (Color.BLACK).getRGB()) {
                pixel = (Color.RED).getRGB();
            }
            biOut.setRGB(x, y, pixel);
        }
      }

      currImage = DStave.toImage(biOut);
    }
}
```

Ουσιαστικά αυτό που κάνει το παραπάνω κομμάτι κώδικα είναι να χρησιμοποιεί τις δύο μεθόδους που δημιουργήσαμε νωρίτερα. Αν η εικόνα δεν είναι null και είναι επιλεγμένη από το χρήστη, τότε δημιουργούμε ένα νέο BufferedImage αντικείμενο «ζωγραφίζοντας» πάνω του το περιεχόμενο της αρχικής εικόνας όπως έχουμε περιγράψει παραπάνω. Κατόπιν ανακαλύπτουμε το width και το height της νέας εικόνας και σαρώνοντας τα pixel της εικόνας αυτής ένα-ένα, συγκρίνουμε το RGB (Red-Green-Blue) με το RGB του μαύρου χρώματος (Color.BLACK) και αντικαθιστούμε τα pixels που είναι μαύρου χρώματος με κόκκινα pixels.

Όταν η διαδικασία ολοκληρωθεί, τοποθετούμε ξανά την εικόνα από την BufferedImage στη μεταβλητή currImage. Τώρα η εντολή:

```
g.drawImage(currImage, totalBeatWidth, pitchTempPos, this);
```

θα εμφανίσει στην οθόνη την κόκκινη εκδοχή της αντίστοιχης εικόνας για την επιλεγμένη μόνο νότα της παρτιτούρας, όπως φαίνεται στο Σχήμα 6-25.



Σχήμα 6-25: Απεικόνιση επιλεγμένης νότας (αλλαγή χρώματος).

6.7 Διάδραση ανάμεσα στην παρτιτούρα και την εικονική κιθάρα

Στο αρχείο DGrandStave στην προηγούμενη ενότητα έχουμε υλοποιήσει όταν κάποια νότα επιλέγεται στην παρτιτούρα να χρωματίζεται με κόκκινο χρώμα. Σε αυτό ακριβώς το σημείο του κώδικα γνωρίζουμε ποια νότα έχει επιλεγεί. Προσθέτουμε λοιπόν μια γραμμή κώδικα :

```
// Aimilia Grigorakaki - Euthimiou Michalitsa -> draw note/rest
if (currImage != null) {
   if (i==getSelectedNote()) {
      Note selected_note = phrase.getNote(i);//.getPitch();
      GuitarNeck.getInstance().setNote(selected_note.getPitch());
```

Η τελευταία γραμμή θα εξυπηρετήσει ώστε στην εικονική κιθάρα να εμφανιστεί αμέσως η αντίστοιχη νότα στο αντίστοιχο τάστο της κιθάρας. Για να γίνει όμως αυτό, πρέπει πρώτα να εξηγήσουμε τις απαραίτητες αλλαγές στον κώδικα του κεφαλαίου 7.5 GuitarNeck.java.

Η πρώτη αλλαγή είναι η μετατροπή του αρχείου αυτού σε Singleton. Singleton [27] είναι ένα Design Pattern [28] όπου αναγκάζουμε την εφαρμογή να διατηρεί MONO ΕΝΑ instance του αντικειμένου αυτού. Ενώ όλα τα αντικείμενα της Java μπορούν να αρχικοποιηθούν πολλές φορές π.χ.

```
JButton button1 = new JButton();
JButton button2 = new JButton();
```

Τα Singleton αντικείμενα αρχικοποιούνται μόνο μια φορά, την πρώτη φορά που τα αρχικοποιούμε. Πως το πετυχαίνουμε αυτό? Κάνουμε τον constructor του αντικειμένου private, δηλαδή ένα αντικείμενο τέτοιου τύπου μπορεί να αρχικοποιηθεί μόνο από τον ίδιο τον εαυτό του.

Δηλαδή προγραμματιστικά δεν μπορούμε να κάνουμε το

JButton button1 = new JButton();

επειδή το 'new JButton()' μέρος της παραπάνω εντολής κάνει χρήση του constructor του αντικειμένου, που πλέον το μετατρέψαμε σε private.

Πώς θα αρχικοποιήσουμε όμως το αντικείμενο? Δημιουργούμε μια μέθοδο με το όνομα getInstance() η οποία αρχικοποιεί το αντικείμενο `GuitarNeck`. Παρακάτω φαίνεται ο απαραίτητος κώδικας:

```
private static GuitarNeck reference;

public static GuitarNeck getInstance() {
      if (reference==null) {
            reference = new GuitarNeck();
      }
      return reference;
}
```

Έχουμε λοιπόν μια μεταβλητή με το όνομα *reference*. Όταν καλούμε τη μέθοδο getInstance() την πρώτη φορά το *reference* είναι null οπότε θα αρχικοποιηθεί μέσω του constructor το αντικείμενο. Αυτό μπορεί να γίνει και ας είναι ο constructor private επειδή ο παραπάνω κώδικας βρίσκεται μέσα στο αρχείο GuitarNeck.java

Το private δηλαδή «κρύβει» κάτι από όλα τα αντικείμενα εκτός από τον εαυτό του. Για να συνεχίσουμε, την δεύτερη, τρίτη κ.ο.κ. φορά που ο χρήστης καλεί τη μέθοδο getInstance, το *reference* δεν είναι πλέον null. Υπάρχει instance του αντικειμένου και το επιστρέφουμε. Το αντικείμενο GuitarNeck με αυτόν τον τρόπο υπάρχει καμία, ή μια φορά μόνο στη μνήμη.

Αυτή η μέθοδος ήταν μια από τις πολλές που θα μπορούσαμε να χρησιμοποιήσουμε για να καταφέρουμε να υλοποιήσουμε μια σύνδεση ανάμεσα στην εικονική κιθάρα και στην παρτιτούρα. Ένας άλλος τρόπος θα ήταν να χρησιμοποιούσαμε events. Όταν δηλαδή επιλεγόταν μια νότα θα κάναμε fire ένα νέο event. Το GuitarNeck από την άλλη, θα έκανε register σαν listener για events και «ακούγοντας» το νέο αυτό event θα αντιδρούσε κατάλληλα.

Επειδή όμως οι παρτιτούρες είναι πολλές ενώ η εικονική κιθάρα μια και μοναδική η λύση που επιλέξαμε με τη μετατροπή του αντικειμένου σε Singleton καλύπτει εύκολα τις ανάγκες της εφαρμογής.

Ο κώδικας του αντικειμένου GuitarNeck χρειάζεται μια νέα μέθοδο η οποία θα δέχεται σαν είσοδο το pitch της επιλεγμένης νότας της παρτιτούρας και θα εμφανίζει τις αντίστοιχες νότες στην εικονική κιθάρα. Η μέθοδος αυτή θα ονομάζεται setNote και έχει τον παρακάτω κώδικα:

```
public void setNote(int pitch) {
      clearAll();
      int fret_1 = (pitch-77); //5
      int fret_2 = (pitch-72); //4
      int fret_3 = (pitch-68); //5
      int fret_4 = (pitch-63); //5
      int fret_5 = (pitch-58); //5
      int fret_6 = (pitch-53);

      if (fret_1>=0 & fret_1<=21) {
            fret_buttons[0][fret_1].setSelected(true);
      }
      if (fret_2>=0 & fret_2<=21) {
            fret_buttons[1][fret_2].setSelected(true);
      }
```

```
            if (fret_3>=0 & fret_3<=21) {
                  fret_buttons[2][fret_3].setSelected(true);
            }
            if (fret_4>=0 & fret_4<=21) {
                  fret_buttons[3][fret_4].setSelected(true);
            }
            if (fret_5>=0 & fret_5<=21) {
                  fret_buttons[4][fret_5].setSelected(true);
            }
            if (fret_6>=0 & fret_6<=21) {
                  fret_buttons[5][fret_6].setSelected(true);
            }

      }
```

Ο παραπάνω κώδικα πρώτα από όλα καλεί τη νέα μέθοδο `clearAll();`

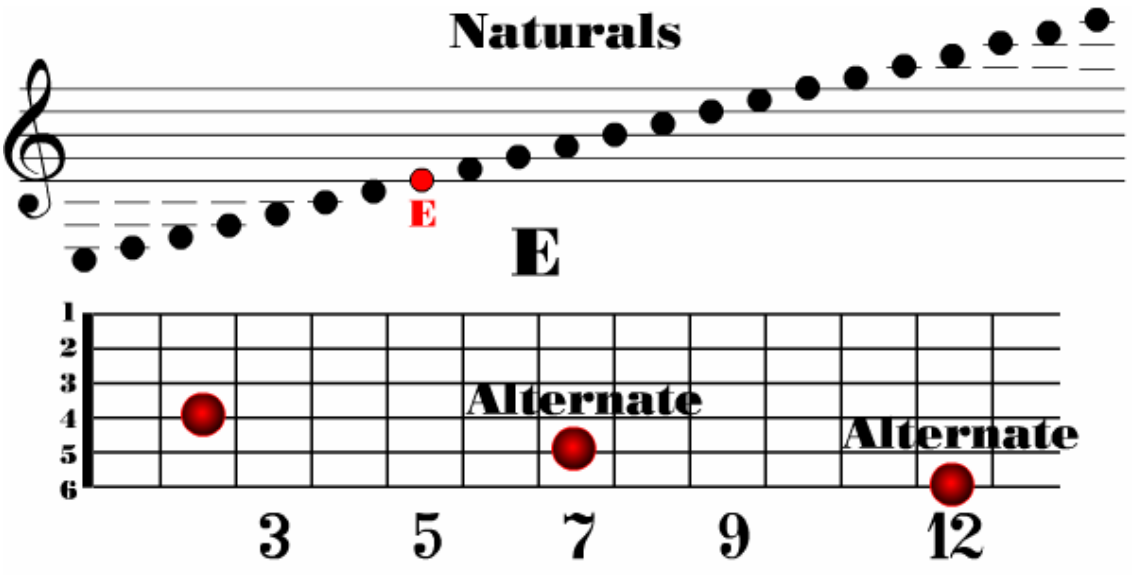
```
      public void clearAll() {
            for (int string=0; string <= 5; string++) {
                  for (int i=0; i<fret_buttons[0].length; i++) {
                        fret_buttons[string][i].setSelected(false);
                  }
            }
      }
```

Η οποία αναλαμβάνει να «καθαρίσει» την εικονική κιθάρα από τις ενεργές νότες.

Αφού τρέξει η μέθοδος αυτή η εικονική κιθάρα είναι έτοιμη να απεικονίσει τη νότα της παρτιτούρας. Όταν ο χρήστης επιλέγει μια νότα στην παρτιτούρα τρέχει η εντολή

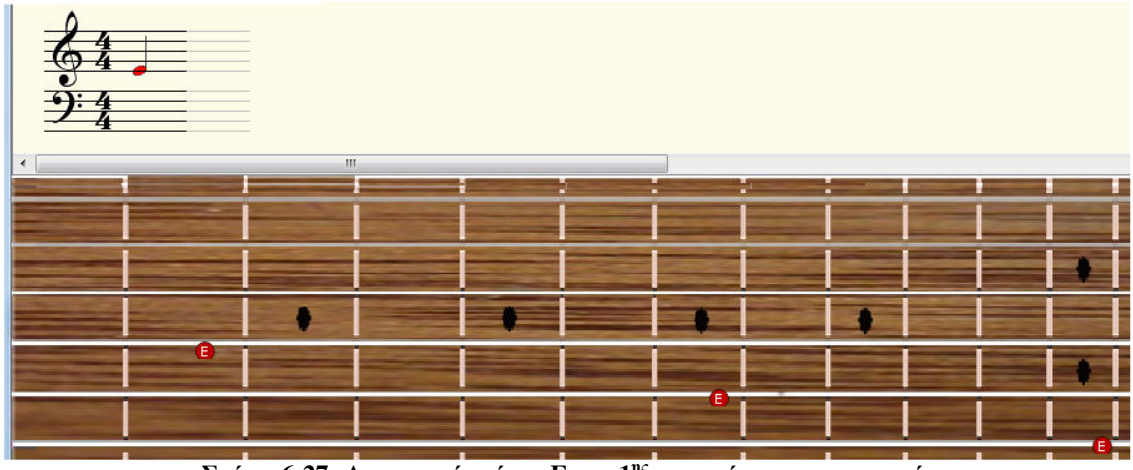
```
GuitarNeck.getInstance().setNote(selected_note.getPitch());
```

Έχουμε σαν παράμετρο λοιπόν το pitch της νότας και θέλουμε να ανακαλύψουμε τη θέση της κιθάρας που αντιστοιχεί στη νότα αυτή. Στην κιθάρα όμως οι νότες της παρτιτούρας αντιστοιχούν σε αρκετά πιθανά σημεία, για παράδειγμα χρησιμοποιώντας ένα on-line εργαλείο [29] βλέπουμε ότι στη νότα Ε στην πρώτη γραμμή του πενταγράμμου στο μουσικό κλειδί ΣΟΛ αντιστοιχεί το δεύτερο τάστο της $4^{ης}$ χορδής της κιθάρας καθώς και το $7^{ο}$ της $5^{ης}$ χορδής και το $12^{ο}$ της $6^{ης}$ χορδής (Σχήμα 6-26).



Σχήμα 6-26: Αντιστοιχία νότας Ε της 1η γραμμής στην ταστιέρα της κιθάρας.

Το ίδιο θέλουμε να πετύχουμε από την εφαρμογή μας. Έτσι όταν στην παρτιτούρα επιλέγεται η νότα Ε όπως φαίνεται στο Σχήμα 6-27, θέλουμε να εμφανίζεται η αντίστοιχη νότα στο μπράτσο της κιθάρας.



Σχήμα 6-27: Αντιστοιχία νότας Ε της 1ης γραμμής στην εφαρμογή.

Το παραπάνω επιτεύχθηκε με τη χρήση της γνώσης του pitch της νότας. Όταν μια νότα έχει για pitch την τιμή 76 τότε αντιστοιχεί στην 1η χορδή της κιθάρας. Με τιμή 77 αντιστοιχεί στο πρώτο τάστο της 1ης χορδής.

Ακολουθώντας τους μαθηματικούς κανόνες που συνδέουν τις νότες με τις αντίστοιχες χορδές και τα τάστα καταλήξαμε στον απλό αλγόριθμο:

```
int fret_1 = (pitch-77); //5
int fret_2 = (pitch-72); //4
int fret_3 = (pitch-68); //5
int fret_4 = (pitch-63); //5
int fret_5 = (pitch-58); //5
int fret_6 = (pitch-53);
```

Η θέση του τάστου (frets) που πρέπει να χρησιμοποιήσει ο καλλιτέχνης μουσικός για να παίξει μια νότα μπορεί να βρεθεί με μια απλή αφαίρεση. Η αφαίρεση ανακαλύπτει τη θέση αυτή. Αν η αφαίρεση επιστρέψει την τιμή 0 τότε μιλάμε για την ανοιχτή χορδή. Αν η τιμή είναι μικρότερη ή ίση με το 22 τότε υπάρχει θέση στη χορδή και η νότα εμφανίζεται στη θέση αυτή.

Υπάρχουν νότες που μπορούν να παίχτουν σε περισσότερες από μια χορδές της κιθάρας. Στην περίπτωση αυτή μια από τις θέσεις αυτές είναι η σύνηθες ενώ οι άλλες θεωρούνται «alternatives» ή εναλλακτικές.

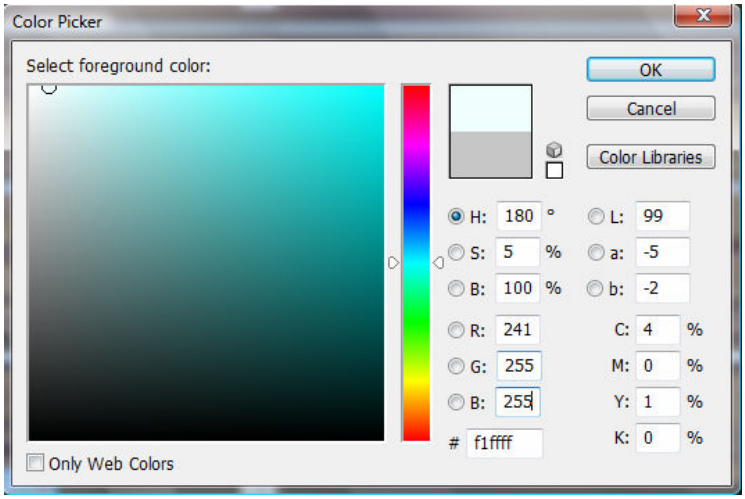
Στην εφαρμογή αυτή φαίνονται όλες οι πιθανές θέσεις στην κιθάρα όταν επιλέγονται στην παρτιτούρα.

Ένα θέμα που δεν έχει αντιμετωπιστεί μέχρι τώρα είναι οι ανοιχτές χορδές, και η υλοποίηση που επιλέχθηκε θα παρουσιαστεί στο επόμενο κεφάλαιο.

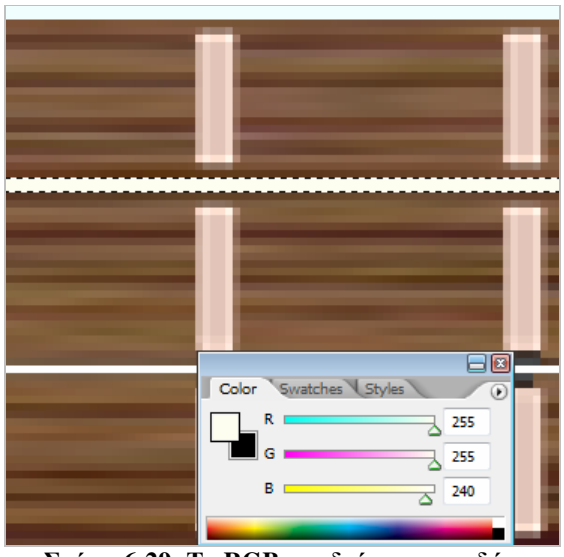
6.8 Ανοιχτές χορδές

Για τις «ανοιχτές» χορδές, όταν δηλαδή η χορδή πρέπει να παίξει χωρίς τη χρήση τάστου ένας πιο έξυπνος τρόπος υλοποίησης χρειαζόταν.

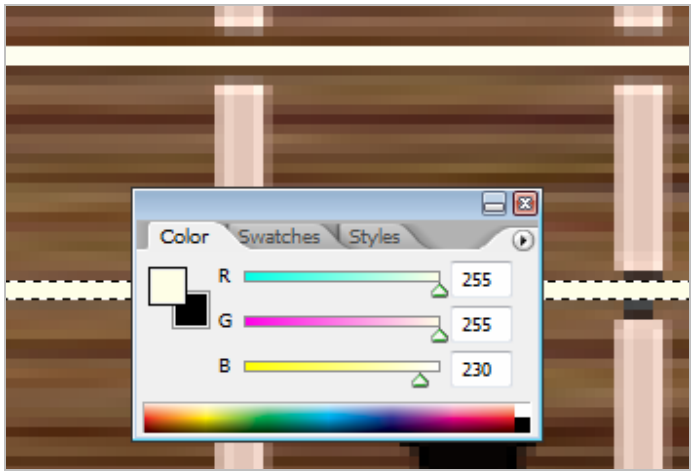
Η υλοποίηση που επιλέχθηκε είναι η παρακάτω: Με χρήση του PhotoShop επεξεργαστήκαμε την εικόνα του μπράτσου της κιθάρας και επιλέξαμε συγκεκριμένα RGB χρωμάτων για κάθε χορδή ξεχωριστά.



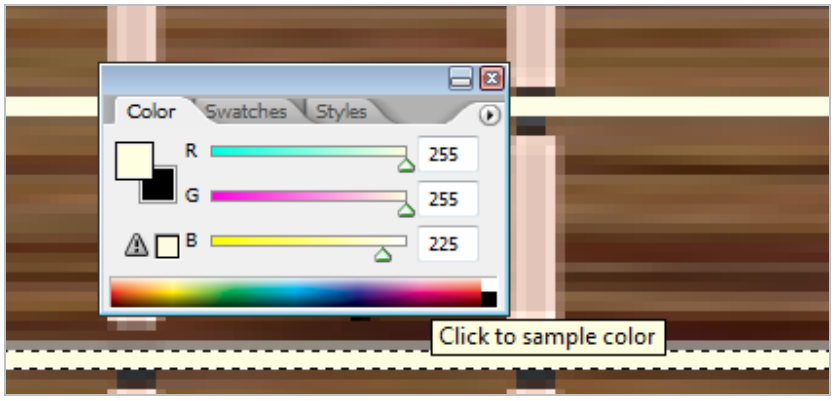
Σχήμα 6-28: Το RGB της πρώτης χορδής.



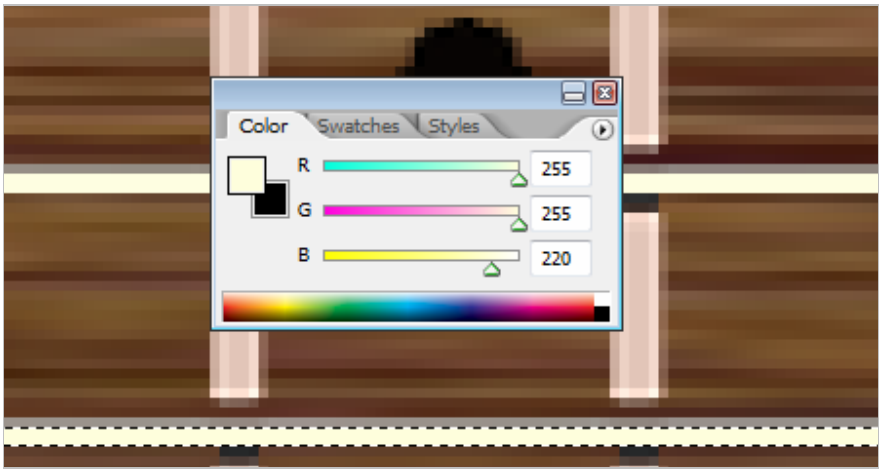
Σχήμα 6-29: Το RGB της δεύτερης χορδής.



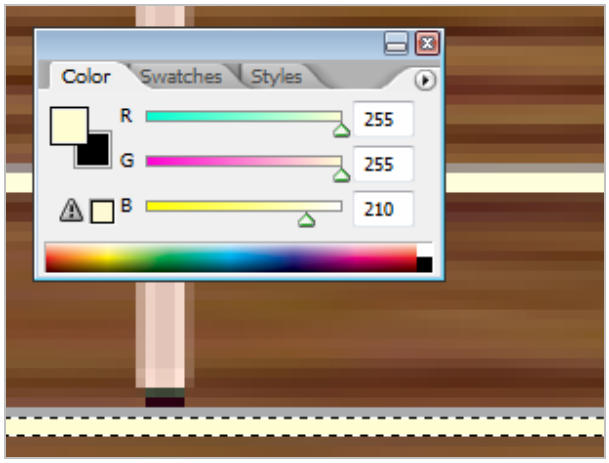
Σχήμα 6-30: Το RGB της τρίτης χορδής.



Σχήμα 6-31: Το RGB της τέταρτης χορδής.



Σχήμα 6-32: Το RGB της πέμπτης χορδής.

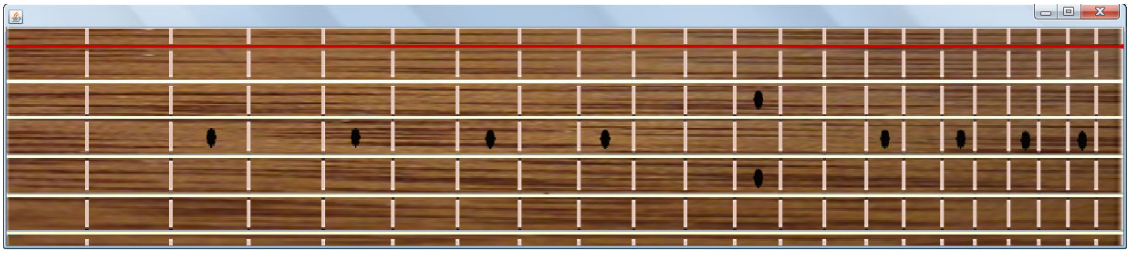


Σχήμα 6-33: Το RGB της έκτης χορδής.

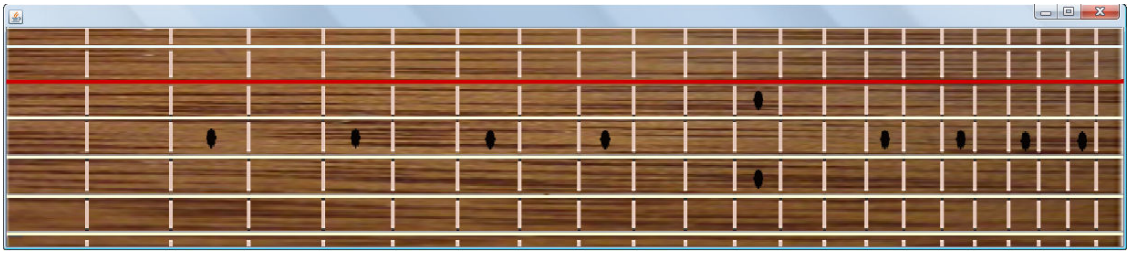
Έτσι η κάθε χορδή είναι ένα συγκεκριμένο RGB:

1η χορδή → 241,255,255
2η χορδή → 255,255,240
3η χορδή → 255,255,230
4η χορδή → 255,255,225
5η χορδή → 255,255,220
6η χορδή → 255,255,210

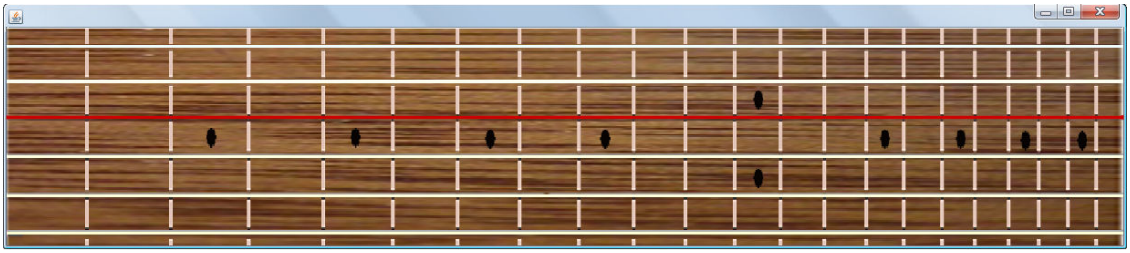
Χρησιμοποιώντας τις μεθόδους για color replacement που υλοποιήσαμε στο κεφάλαιο 7.6 μπορούμε πλέον να χρωματίσουμε τις χορδές επειδή απεικονίζονται με διακριτό RGB στην εικόνα με ένα έντονο κόκκινο χρώμα για να καταλάβει ο χρήστης ότι πρέπει να παίξει απλά τη χορδή.



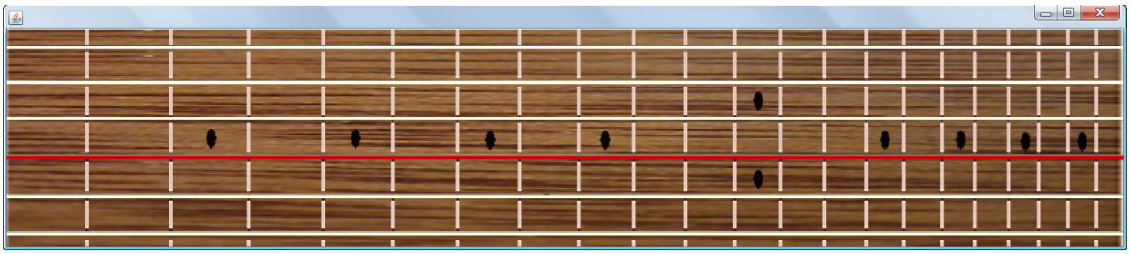
Σχήμα 6-34: Απεικόνιση 1ης χορδής ανοιχτής.



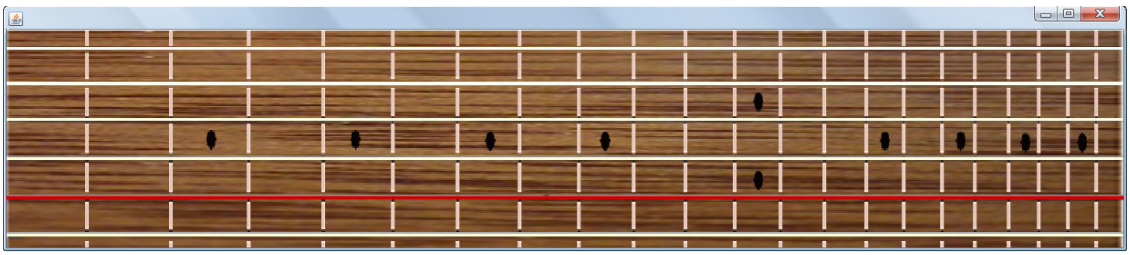
Σχήμα 6-35: Απεικόνιση 2ης χορδής ανοιχτής.



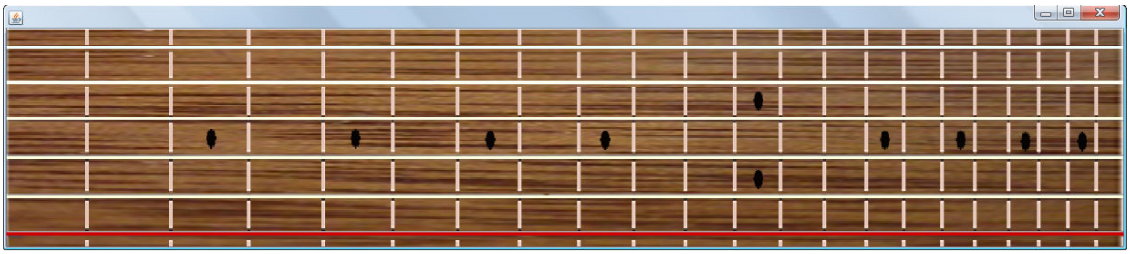
Σχήμα 6-36: Απεικόνιση 3ης χορδής ανοιχτής.



Σχήμα 6-37: Απεικόνιση 4ης χορδής ανοιχτής.



Σχήμα 6-38: Απεικόνιση 5ης χορδής ανοιχτής.



Σχήμα 6-39: Απεικόνιση 6ης χορδής ανοιχτής.

Εφόσον η δοκιμή έγινε με επιτυχία με πρόχειρο κώδικα ήρθε η στιγμή να υλοποιηθεί με δημιουργία μεθόδων ώστε να μπορούμε εύκολα να εμφανίσουμε όποια χορδή θέλουμε.

Στην αρχή του αντικειμένου τοποθετούμε τα χρώματα που τοποθετήσαμε στις χορδές.

```
Color color_1st = new Color(241,255,255);
Color color_2nd = new Color(255,255,240);
Color color_3rd = new Color(255,255,230);
Color color_4th = new Color(255,255,225);
Color color_5th = new Color(255,255,220);
Color color_6th = new Color(255,255,210);
```

Κατόπιν δημιουργούμε μέθοδο η οποία κάνει την απαραίτητη χρωματική αλλαγή. Παίρνει σαν παράμετρο τον αριθμό της χορδής (από 1 ως 6), εντοπίζει το χρώμα της χορδής, κάνει τη χρωματική αλλαγή και καλεί την repaint εντολή για να ανανεωθεί το γραφικό περιβάλλον της εφαρμογής.

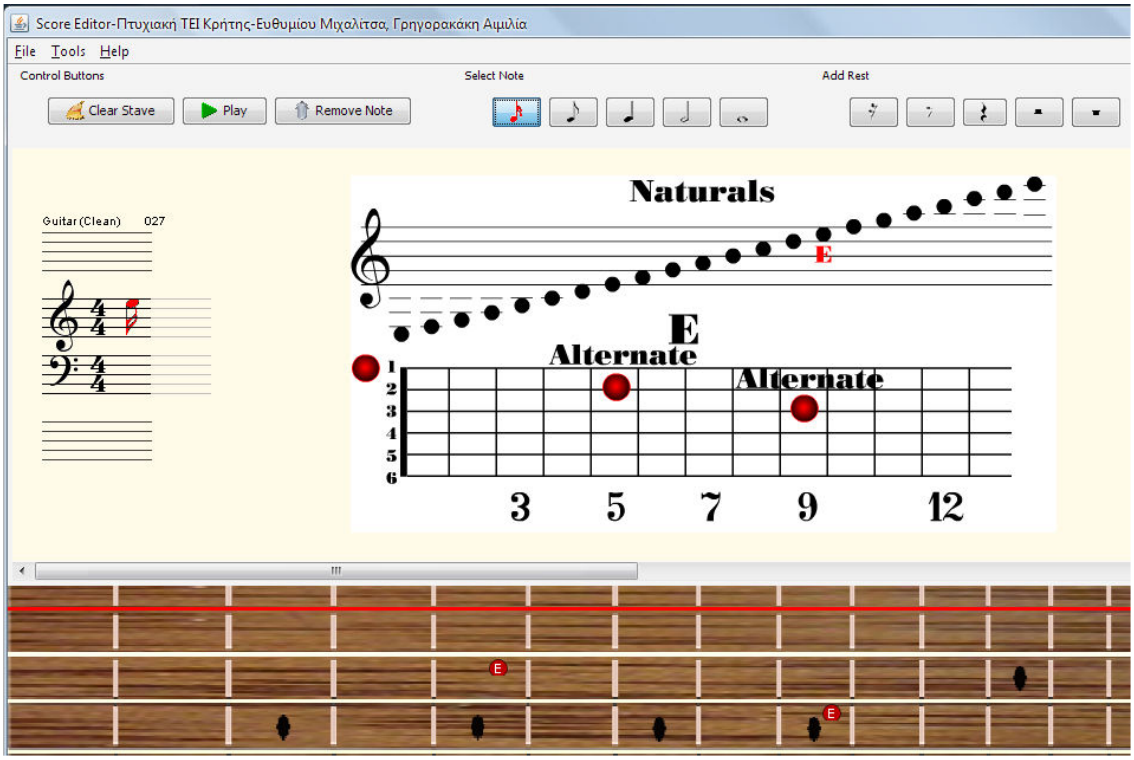
```
/**
 * Switch the color of the string
 * @param string values 1 to 6
 */
private void switchColor(int string) {
      Color new_color;

      switch (string) {
            case 1: new_color = color_1st; break;
            case 2: new_color = color_2nd; break;
            case 3: new_color = color_3rd; break;
            case 4: new_color = color_4th; break;
            case 5: new_color = color_5th; break;
            case 6: new_color = color_6th; break;
            default: new_color = Color.black;
      }

      BufferedImage bi = DStave.toBufferedImage(ii.getImage());
      int w = bi.getWidth();
    int h = bi.getHeight();
    int pixel;
     BufferedImage biOut = new BufferedImage(w, h,
     bi.TYPE_INT_ARGB_PRE);

    for (int x = 0; x < w; x++) {
         for (int y = 0; y < h; y++) {
              pixel = bi.getRGB(x, y);


              if (pixel == (new_color).getRGB()) {
                pixel = (Color.RED).getRGB();
              }
                  biOut.setRGB(x, y, pixel);
          }
    }

    ii = new ImageIcon(DStave.toImage(biOut));

    repaint();
}
```

Στη συνέχεια πρέπει να προσέξουμε τη μέθοδο clearAll. Μέχρι τώρα «καθάριζε» την εικονική κιθάρα από τις νότες. Τώρα η μέθοδος αυτή θα πρέπει να την «καθαρίζει» και από τις επιλεγμένες-κοκκινισμένες χορδές. Έτσι η μέθοδος μετατρέπεται:

```
public void clearAll() {
      for (int string=0; string <= 5; string++) {
            for (int i=0; i<fret_buttons[0].length; i++) {
                  fret_buttons[string][i].setSelected(false);
            }
      }
```

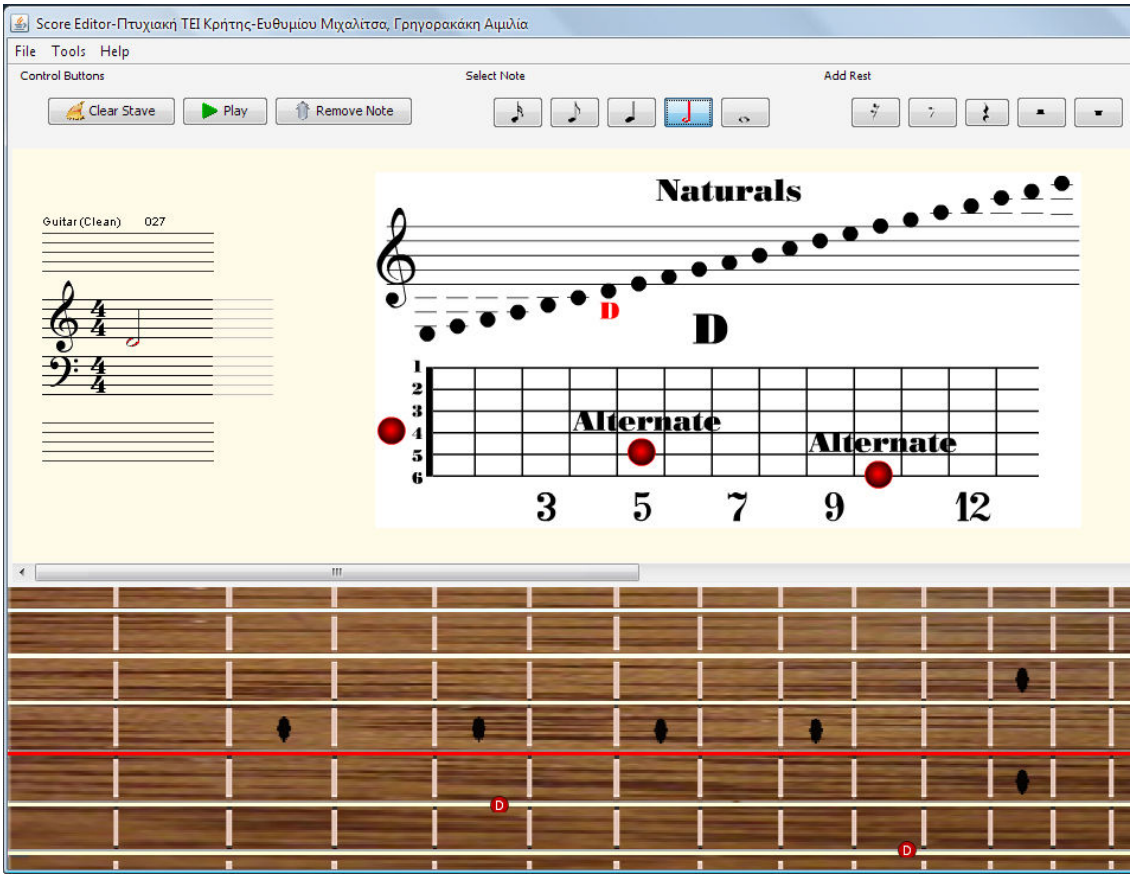
```
        String imgLocation2 = "images/GuitarNeck.png";
        URL imageURL = GuitarNeck.class.getResource(imgLocation2);
        ii = new ImageIcon(imageURL, "altText");
        repaint();
}
```

Σε αυτό το σημείο ολοκληρώνεται η φάση της υλοποίησης. Μάλιστα χρειάστηκαν μόλις 300 γραμμές κώδικα για την υλοποίηση τόσο της εικονικής κιθάρας όσο και της διεπαφής της με την παρτιτούρα. Παρακάτω παρουσιάζονται παραδείγματα χρήσης της εφαρμογής, όπως αυτή υλοποιήθηκε.



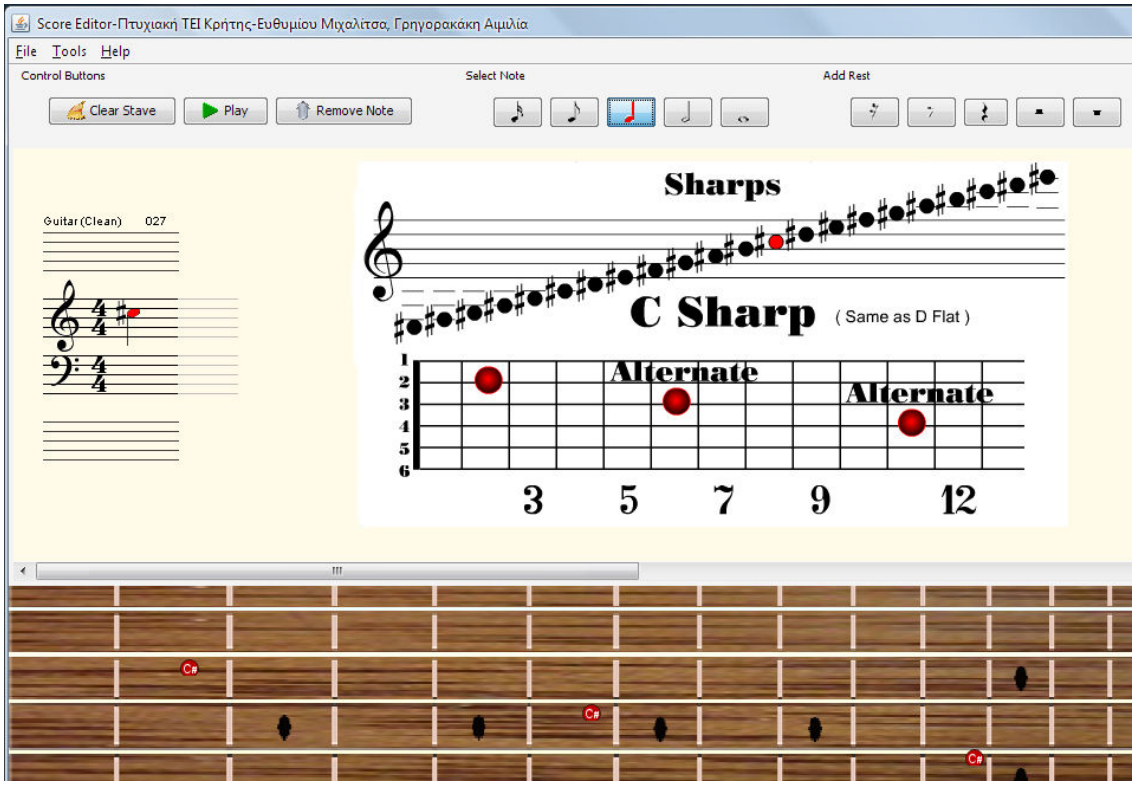
Σχήμα 6-40: Αντιστοιχία νότας Ε 4ου διαστήματος στην εφαρμογή.

Στο Σχήμα 6-40 φαίνεται η αντιστοιχία του προγράμματος που αναπτύξαμε με τη θεωρία. Η νότα Ε μπορεί να παιχτεί σαν «ανοιχτή πρώτη χορδή» ή στην $5^{η}$ θέση της δεύτερης χορδής ή στην $9^{η}$ θέση της $3^{ης}$ χορδής.



Σχήμα 6-41: Αντιστοιχία νότας D στην εφαρμογή.

Στο Σχήμα 6-41 απεικονίζεται η αντιστοιχία της νότας D όπου η σωστή θέση της στην κιθάρα είναι η ανοιχτή 3^{η} χορδή ή η 5^{η} θέση της 5^{ης} χορδής ή η 10^{η} θέση της 6^{ης} χορδής.



Σχήμα 6-42: Αντιστοιχία νότας C - sharp στην εφαρμογή.

Επίσης εμφανίζονται σωστά και οι νότες με σημεία αλλοίωσης όπως π.χ. **η δίεση (#)** που ανεβάζει το φθόγγο ένα ημιτόνιο. Στο σχήμα 6-42 εμφανίζονται οι θέσεις που αντιστοιχούν στην C-Sharp.

6.9 Δημιουργία Ant Script

Για να μπορεί η εφαρμογή να πακεταριστεί όμορφα θα πρέπει όλα τα .class αρχεία μετά το compilation να μπουν σε ένα αρχείο .JAR Για να γίνει αυτό πρέπει πρώτα από όλα να δημιουργηθεί ένα MANIFEST αρχείο [25]. Κατόπιν μελέτης το τελικό manifest.txt ακολουθεί.

```
Manifest-Version: 2.0
Main-Class: diamouses.ui.scoreEditor.Main
Class-Path: jmusic.jar
Created-By: 1.5 (Sun Microsystems Inc.)

Name:
Specification-Title: @Name@
Specification-Version: @version@
Specification-Vendor: @vendor@
Implementation-Vendor: Aimilia
Implementation-Version: @version@
Implementation-Title: ScoreEditor
```

Το παραπάνω αρχείο περιγράφει πού θα βρεθεί η κύρια κλάση της εφαρμογής (δηλαδή στο `diamouses.ui.scoreEditor.Main`) και ποιες βοηθητικές βιβλιοθήκες χρησιμοποιούνται και απαιτούνται για να τρέξει η εφαρμογή (στην εφαρμογή της πτυχιακής αυτής εργασίας, απαραίτητη είναι η ύπαρξη και χρήση της βιβλιοθήκης jmusic.jar).

Εφόσον δημιουργηθεί το MANIFEST αρχείο, ένα ακόμα θέμα είναι ότι η εφαρμογή ακολουθείται από ορισμένες εικόνες που πρέπει και αυτές να τοποθετηθούν στο .JAR Και ενώ η εφαρμογή φόρτωνε αρχικά τις εικόνες κανονικά, όταν τοποθετούνταν μέσα σε .JAR αρχείο δεν μπορούσε να τις εντοπίσει. Η λύση δόθηκε με τη χρήση του getClass().getClassLoader().getResource():

```
public BufferedImage loadImage(String name) {
    String imgFileName = "images/"+name;
    URL url = getClass().getClassLoader().getResource(imgFileName);
    BufferedImage img = null;
    try {
        img = ImageIO.read(url);
    } catch (Exception e) {
        System.err.println(e.toString());
    }
    return img;
}
```

Όταν δηλαδή η εφαρμογή αποτελείται από .class αρχεία τότε μπορούμε να χρησιμοποιήσουμε το filesystem ευθέως π.χ. URL url = “images/image.gif”. Όταν όμως πακεταριστούν τα δεδομένα μέσα σε ένα JAR αρχείο, που στην πράξη δεν είναι τίποτα παραπάνω από ένα συμπιεσμένο αρχείο τύπου RAR, πρέπει να καλέσουμε την getResource μέθοδο για να αποκτήσουμε πρόσβαση στα αρχεία που περικλείονται στο αρχείο JAR.

Εφόσον οι παραπάνω προϋποθέσεις εκπληρώθηκαν (δηλαδή η δημιουργία του MANIFEST αρχείου, και οι απαραίτητες μετατροπές στον κώδικα της εφαρμογής) με την χρήση μιας σειράς μακροσκελών εντολών μπορούμε να κάνουμε compile τα αρχεία της εργασίας και να τα τοποθετήσουμε μαζί με τα υπόλοιπα αρχεία (εικόνες κτλ) σε ένα .JAR.

Για να αυτοματοποιηθεί όμως η παραπάνω διαδικασία κατασκευής του αρχείου .JAR αποφασίστηκε να χρησιμοποιηθεί ένα ant script file [26]. Ονομάζεται build.xml και βρίσκεται στο root κατάλογο του project.

Το ANT αποτελεί ένα open source project του Apache Foundation και είναι ένα εργαλείο που βοηθάει στην αυτοματοποίηση της διαδικασίας BUILD και packaging εφαρμογών java.

Για να χρησιμοποιήσει κανείς το ANT αφού το εγκαταστήσει και το τοποθετήσει στο PATH του λειτουργικού συστήματος πρέπει να δημιουργήσει ένα αρχείο XML, το build.xml.

Το XML αυτό αρχείο αποτελείται από **tasks**, κάθε ένα από τα οποία κάνει μια συγκεκριμένη εργασία. Δημιουργήθηκε λοιπόν ένα build.xml αρχείο με τα tasks που παρουσιάζονται στον Πίνακα 6-1.

TASK NAME	Περιγραφή – Επεξήγηση
variables	Αρχικοποιεί μεταβλητές που θα χρησιμοποιηθούν στη συνέχεια της διαδικασίας του build, όπως βοηθητικές βιβλιοθήκες, τοποθεσία φακέλων και εικόνων.
init	Δημιουργεί το φάκελο build/classes όπου θα τοποθετηθούν τα .class (τα compiled αρχεία), το φάκελο build/jars όπου θα τοποθετηθούν τα .JAR αρχεία και αντιγράφει τις φωτογραφίες (images) από το source directory στο destination directory
compile	Κάνει compile τα αρχεία .java της εφαρμογής
jar	Πακετάρει τα αρχεία .class το Manifest.txt και τις εικόνες σε ένα αρχείο ScoreEditor.jar
clean	Σβήνει τους φακέλους /build/classes και /build/jars «καθαρίζοντας» το περιβάλλον για δημιουργία νέας έκδοσης της εφαρμογής
run	Εκτελεί την εφαρμογή

Πίνακας 6-1: Λίστα των tasks που υλοποιούνται στο αρχείο build.xml.

Τέλος πρέπει να ειπωθεί ότι ορισμένα tasks εξαρτούνται (depend) από άλλα tasks. Δηλαδή για να τρέξει το jar-task το οποίο εξαρτάται από το init-task και το compile-task, τρέχουν πρώτα τα απαραίτητα task και κατόπιν το jar-task.

```
<?xml version="1.0"?>
<!-- ======================================================================= -->
<!-- A Buildfile                                                             -->
<!-- ======================================================================= -->
<project name="ScoreEditor" default="jar" basedir=".">

 <target name="variables">
   <tstamp/>

   <property name="Name"              value="ScoreEditor"/>

   <property name="src.dir" value="${basedir}${file.separator}src"/>

   <property name="lib.dir"          value="${basedir}${file.separator}libs"/>
   <property name="build.dir"        value="${basedir}${file.separator}build"/>
   <property name="build.classes"value="${build.dir}${file.separator}classes"/>
   <property name="build.jar.dir" value="${build.dir}${file.separator}jars"/>
   <property name="build.jar"
value="${build.jar.dir}${file.separator}${Name}.jar"/>

<property name="build.classpath"
value=".:${lib.dir}${file.separator}jmusic.jar"/>

 <property name="images.src"
value="${src.dir}${file.separator}diamouses${file.separator}ui${file.sepa
rator}scoreEditor${file.separator}images"/>

<property name="images.dest"
value="${build.classes}${file.separator}diamouses${file.separator}ui${fil
e.separator}scoreEditor${file.separator}images"/>

   <property name="packages"   value="*"/>
</target>
<!-- ======================================================================= -->
<!-- Create required directories                                             -->
<!-- ======================================================================= -->
    <target name="init" depends="variables">
        <!-- Prepare necessary directories -->
        <mkdir dir="${build.dir}"/>
        <mkdir dir="${build.classes}"/>
        <mkdir dir="${build.jar.dir}"/>
        <mkdir dir="${images.dest}"/>

        <!-- Copy all the images to the build directory -->
        <copy todir="${images.dest}">
            <fileset dir="${images.src}" casesensitive="yes">
                <include name="**/*.*"/>
            </fileset>
        </copy>
</target>
```

```
<!-- ================================================================= -->
<!-- Compiles the source code                                          -->
<!-- ================================================================= -->
    <target name="compile" depends="init" description="Compiles source
code">
        <javac encoding="UTF-8"
               srcdir="${src.dir}"
               destdir="${build.classes}"
               classpath="${build.classpath}"
               optimize="on"
                />
    </target>
<!-- ================================================================== -->
<!-- Creates a jar archive                                              -->
<!-- ================================================================= -->
    <target name="jar" depends="init,compile" description="Generates
ScoreEditor.jar" >
<!-- Put everything from ${build.dir} into the ScoreEditor.jar file -->
        <jar
                manifest="${src.dir}/myManifest.txt"
                jarfile="${build.jar.dir}${file.separator}${Name}.jar"
                basedir="${build.classes}"
                includes="**"
                />
        <copy todir="${build.jar.dir}">
            <fileset dir="${src.dir}" casesensitive="yes">
                <include name="**/*.html"/>
            </fileset>
        </copy>
        <copy todir="${build.jar.dir}">
            <fileset dir="${lib.dir}" casesensitive="yes">
                <include name="**/*.jar"/>
            </fileset>
        </copy>
    </target>
<!-- ================================================================= -->
<!-- Cleans up all                                                     -->
<!-- ================================================================= -->
    <target name="clean" depends="init" description="Removes previous
build (classes and jar files)">
        <delete dir="${build.classes}"/>
        <delete dir="${build.jar.dir}"/>
    </target>
<!-- ================================================================= -->
<!-- Run Score Editor                                                  -->
<!-- ================================================================= -->
    <target name="run" depends="init,compile,jar" description="Initiates
ScoreEditor.java">
        <echo message = "Running ScoreEditor"/>
        <java
                        jar="${build.jar.dir}${file.separator}${Name}.jar"
      fork="true"

        >
        </java>
    </target>
</project>
```

Φαίνεται λοιπόν η jar εντολή που παίρνει σαν παράμετρο το myManifest.txt βρίσκει τα .class αρχεία και τις εικόνες στο φάκελο που ενημερώσαμε νωρίτερα και τα πακετάρει όλα στο φάκελο build/jars στο αρχείο ScoreEditor.jar. Στον φάκελο αυτό τοποθετεί και το jmusic.jar μιας και είναι απαραίτητο για την εφαρμογή της πτυχιακής αυτής εργασίας.

Με 'ant –p' βλέπουμε τα tasks που υποστηρίζονται από το script (βλέπε Σχήμα 6-43).

```
D:\Workspace\Score-Editor>ant -p
Buildfile: build.xml

Main targets:

 clean    Removes previous build (classes and jar files)
 compile  Compiles source code
 jar      Generates ScoreEditor.jar
 run      Initiates ScoreEditor.java
Default target: jar
D:\Workspace\Score-Editor>_
```

Σχήμα 6-43: Λίστα με tasks που υποστηρίζει το script.

Με 'ant clean' διαγράφεται το προηγούμενο compilation της εφαρμογής (βλέπε Σχήμα 6-44).

```
D:\Workspace\Score-Editor>ant clean
Buildfile: build.xml

variables:

init:

clean:
   [delete] Deleting directory D:\Workspace\Score-Editor\build\classes
   [delete] Deleting directory D:\Workspace\Score-Editor\build\jars

BUILD SUCCESSFUL
Total time: 0 seconds
```

Σχήμα 6-44: Απεικόνιση εκτέλεσης εντολής 'ant clean'.

Και τέλος με 'ant jar' γίνεται compile η εφαρμογή, αντιγράφονται οι εικόνες, και τοποθετούνται όλα τα απαραίτητα αρχεία μαζί με το myManifest.txt σε αρχείο με όνομα ScoreEditor.jar όπως φαίνεται στο Σχήμα 6-45.

```
D:\Workspace\Score-Editor>ant jar
Buildfile: build.xml

variables:

init:
    [mkdir] Created dir: D:\Workspace\Score-Editor\build\classes
    [mkdir] Created dir: D:\Workspace\Score-Editor\build\jars
    [mkdir] Created dir: D:\Workspace\Score-Editor\build\classes\diamouses\ui\sc
oreEditor\images
     [copy] Copying 113 files to D:\Workspace\Score-Editor\build\classes\diamous
es\ui\scoreEditor\images

compile:
    [javac] Compiling 13 source files to D:\Workspace\Score-Editor\build\classes

    [javac] Note: Some input files use unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.

jar:
      [jar] Building jar: D:\Workspace\Score-Editor\build\jars\ScoreEditor.jar
     [copy] Copying 1 file to D:\Workspace\Score-Editor\build\jars

BUILD SUCCESSFUL
Total time: 5 seconds
D:\Workspace\Score-Editor>
```

Σχήμα 6-45: Απεικόνιση εκτέλεσης εντολής 'ant jar'.

Εφόσον τρέξει το task του ANT η εφαρμογή βρίσκεται πακεταρισμένη στον φάκελο build/jars και μπορεί να τρέξει με την εντολή 'java -jar ScoreEditor.jar' ή με double-click από το λειτουργικό σύστημα.

6.10 Μετατροπή εφαρμογής σε Applet

Για να τρέξει η εφαρμογή μέσα από μια ιστοσελίδα πρέπει να μετατραπεί σε Java Applet. Εφόσον έχουμε ήδη εξηγήσει τη μέθοδο τοποθέτησης των interpreted αρχείων με την κατάληξη .class και των εικόνων σε αρχείο .JAR (κάτι που κρίνεται απαραίτητο ώστε η εφαρμογή να μπορεί να τρέξει σαν Applet) προχωράμε στη συγγραφή του απαραίτητου κώδικα HTML που δίνει τις κατάλληλες οδηγίες στον φυλλομετρητή όπως μέγεθος του Applet καθώς και την starting class:

```
<HTML>
<HEAD>
<TITLE>Εκμάθηση κιθάρας</TITLE>
<META content="MSHTML 5.50.4616.200" name=GENERATOR>
<link rel="stylesheet" href="css/styles.css" type="text/css">
</HEAD>
<BODY>
 <applet code="diamouses.ui.scoreEditor.Main.class"
      codebase="."
      archive="ScoreEditor.jar"
      width="1160" height="600">
   Your browser is completely ignoring the &lt;APPLET&gt; tag!
 </applet>
</center>
</BODY></HTML>
```

Δοκιμάζοντας από τοπικό υπολογιστή να ανοίξουμε το παραπάνω αρχείο το οποίο το τοποθετούμε στον φάκελο με τα δύο JAR αρχεία, το ένα της εφαρμογής μας και το άλλο της βιβλιοθήκης JMusic, το Applet αρνείται να φορτώσει και εμφανίζεται ένα μήνυμα:

java.security.AccessControlException: access denied (java.lang.RuntimePermission)

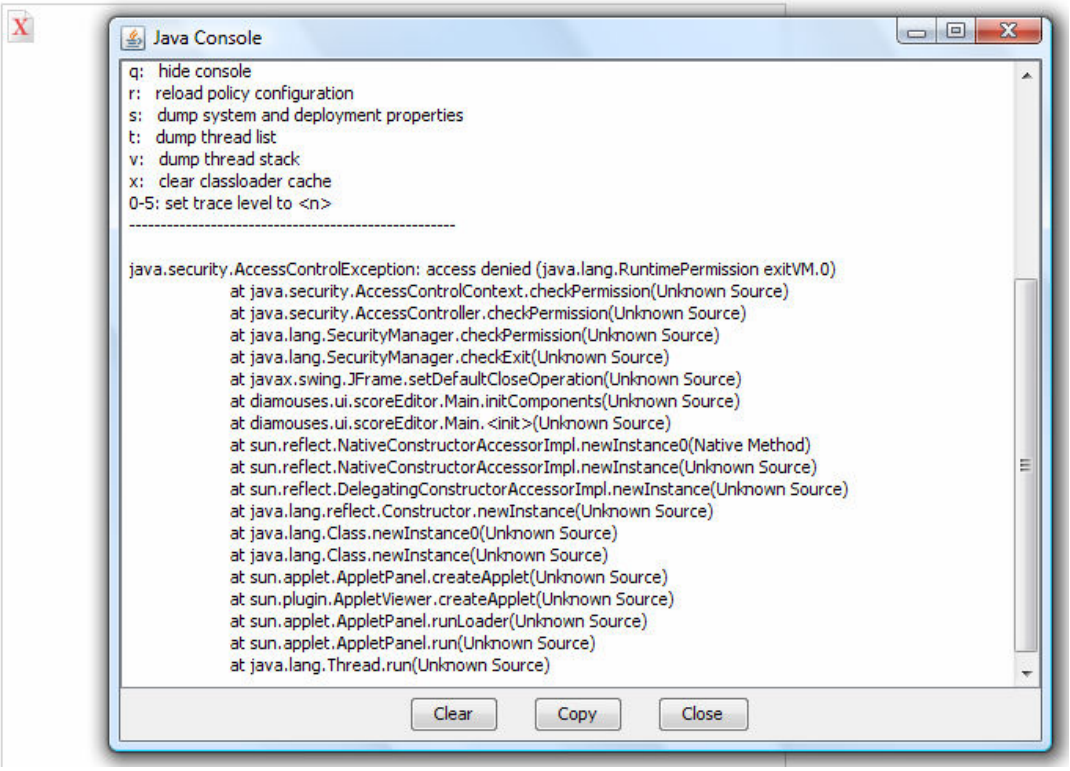
Κοιτάζοντας πιο προσεκτικά εντοπίζουμε στην έβδομη (7) γραμμή του exception ότι το μήνυμα λάθους οφείλεται στη χρήση του JFrame.setDefaultCloseOperation. Βρίσκοντας το σημείο του κώδικα στο οποίο οφείλεται το λάθος καταλαβαίνουμε την αιτία:

```
private void initComponents() {
    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    pack();
}
```

Στα Applets η Java δεν επιτρέπει το κλείσιμο της Java Virtual Machine. Ο λόγος είναι ότι επίδοξοι hackers θα χρησιμοποιούσαν αυτή τη δυνατότητα για να κάνουν DOS attacks (Denial Of Service επιθέσεις σε περιηγητές του Ιστού).

Το άνοιγμα και το κλείσιμο της Java Virtual Machine είναι μια σχετικά χρονοβόρα διαδικασία η οποία καταναλώνει resources (τόσο μνήμη όσο και επεξεργαστή). Ένας κακόβουλος χρήστης θα μπορούσε να γράψει απλά Applets που θα ανοιγόκλειναν την Java Virtual Machine εκατοντάδες φορές το δευτερόλεπτο.

Αυτή η διαδικασία θα «κράσαρε» ακόμα και τα πιο σύγχρονα υπολογιστικά μηχανήματα των χρηστών, που επισκεπτόμενοι τη σελίδα αυτή δεν θα μπορούσαν να αποφύγουν την παγίδα του κακόβουλου χρήστη. Για το σκοπό αυτό απαγορεύτηκε το κλείσιμο της JVM. Στο Σχήμα 6-46 φαίνεται το μήνυμα λάθους.



Σχήμα 6-46: Μήνυμα λάθους της Java Console.

Εναλλακτικά μπορούμε να αντικαταστήσουμε τη γραμμή

```
setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
```

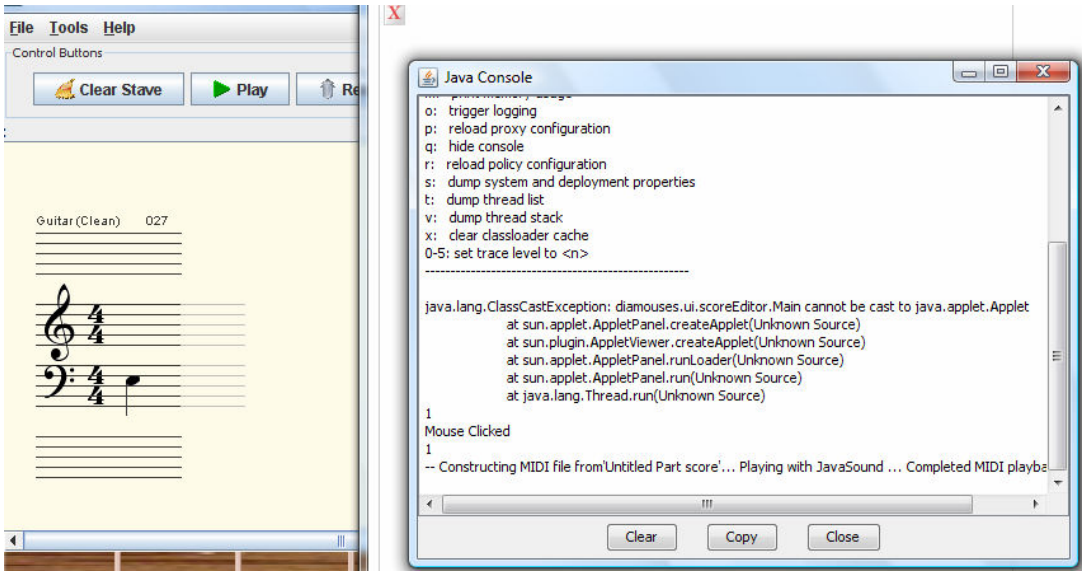
με τη γραμμή

```
setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
```

Κατόπιν ξαναδημιουργούμε το νέο JAR αρχείο που περιέχει τον ανανεωμένο κώδικα.

Αυτή τη φορά η εφαρμογή μας τρέχει σαν Application και στη θέση του Applet βρίσκουμε μήνυμα λάθους που μας λέει:

java.lang.ClassCastException: diamouses.ui.scoreEditor.Main cannot be cast to java.applet.Applet (βλέπε Σχήμα 6-47) .



Σχήμα 6-47: Exception: diamouses.ui.scoreEditor.Main cannot be cast to java.applet.Applet.

Η εφαρμογή χρειάζεται και άλλες αλλαγές για τη μετατροπή της σε Applet. Μέχρι τώρα η Main έκανε extend το JFrame. Αυτό αναγκάζει την εφαρμογή να τρέξει σε νέο παράθυρο.

```
public class Main extends javax.swing.JFrame
```

έτσι η Main θα χρειαστεί πλέον να κάνει extend το JApplet. Ο λόγος είναι ότι επιθυμούμε η εφαρμογή να τρέχει τόσο σαν Application όσο και σαν Applet την ίδια στιγμή. Ο τρόπος που μπορεί να επιτευχθεί αυτό είναι να τοποθετήσουμε τον initialization κώδικα σε μέθοδο με την ονομασία init και να καταργήσουμε τη χρήση του JFrame. Βέβαια αυτό απαιτεί αλλαγές σε πολλά σημεία του κώδικα. Αφού όμως κάναμε τις κατάλληλες τροποποιήσεις τα οφέλη ήταν πολλαπλά.

Ένα ακόμα πρόβλημα που αντιμετωπίσαμε κατά τη μετατροπή του ScoreEditor σε Java Applet ήταν η παράκαμψη των μηχανισμών ασφαλείας του sandbox της Java Virtual Machine που απαγορεύει τη χρήση του JFileChooser για εγγραφή και ανάγνωση αρχείων από τοπικό filesystem του χρήστη στον φυλλομετρητή του οποίου τρέχει το Java Applet.

Όπως είπαμε η JVM για να προστατέψει τους χρήστες περιέχει έναν πολύ αυστηρό μηχανισμό ασφαλείας ειδικά στα Applets για να αποτρέψει την κακόβουλη χρήση τους. Παρέχει όμως μηχανισμούς για τροποποίηση των policy settings παρέχοντας μέσα από πιστοποιητικά ασφαλείας και κατόπιν συναίνεσης του χρήστη priviledge escalation, δηλαδή παροχή αυξημένων δικαιωμάτων σε ένα applet.

Συγκεκριμένα 3 εργαλεία παρέχονται μαζί με το JDK, τα:

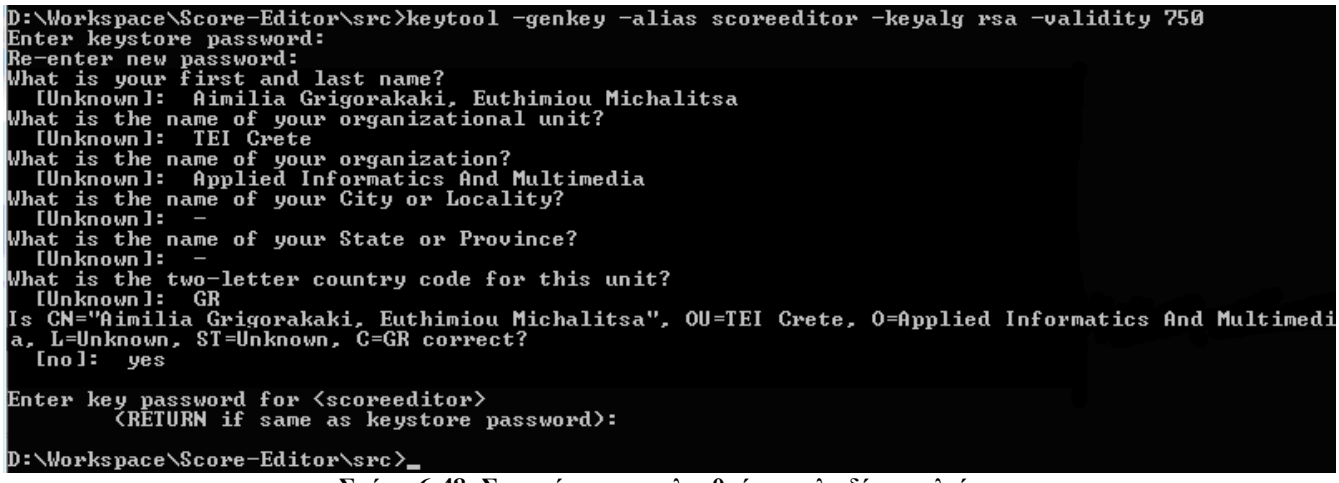
- Keytool: Δημιουργεί και κάνει manage κλειδιά ασφαλείας
- Jarsigner: Υπογράφει αρχεία JAR με κάποιο κλειδί ασφαλείας που έχουμε ήδη δημιουργήσει.
- Policytool: Γραφικό περιβάλλον για διαχείριση πολιτικής ασφαλείας (policy management).

Το πρώτο βήμα λοιπόν είναι να χρησιμοποιήσουμε το keytool και να δημιουργήσουμε ένα κλειδί ασφαλείας. Το εργαλείο αυτό δημιουργεί κλειδιά βασισμένα στο πρωτόκολλο X.509.

Το X.509 είναι ένα πρότυπο κρυπτογράφησης το οποίο σχεδιάστηκε για να παρέχει υποδομή πιστοποίησης. Η πρώτη έκδοση του X.509 δημοσιεύθηκε το 1988, καθιστώντας το την παλαιότερη πρόταση για μια παγκόσμια Υποδομή Δημόσιου Κλειδιού. Σε συνδυασμό με την υποστήριξη του προτύπου από τον ISO και από άλλους οργανισμούς όπως τη Διεθνή Ένωση Τηλεπικοινωνιών (International Telecommunications Union - ITU), το X.509 έχει διεθνώς αναγνωριστεί. Αρκετά χρηματοπιστωτικά ιδρύματα χρησιμοποιούν το X.509 για το πρότυπο ασφαλών συναλλαγών SET (Secure Electronic Transactions). Με τα παραπάνω φαίνεται ότι αποτελεί ένα σοβαρό πρότυπο ασφαλείας και για το λόγο αυτό χρησιμοποιείται από την Sun για πιστοποίηση στα Applets.
Για να δημιουργήσουμε λοιπόν ένα κλειδί ασφαλείας, πληκτρολογούμε την παρακάτω εντολή:

```
keytool -genkey -alias scoreeditor -keyalg rsa -validity 750
```

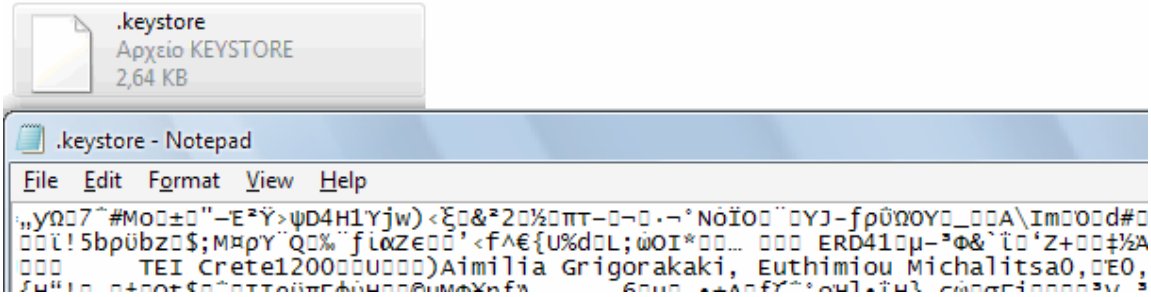
Ζητάμε δηλαδή τη δημιουργία κλειδιού με το alias scoreeditor με χρήση του αλγορίθμου RSA και με διάρκεια ζωής 750 μέρες. Στο Σχήμα 6-48 που ακολουθεί μπορείτε να παρατηρήσετε τα στοιχεία που συμπληρώνουμε και ακολουθούν το κλειδί ασφαλείας.



Σχήμα 6-48: Στοιχεία που ακολουθούν το κλειδί ασφαλείας.

Εφόσον προετοιμάσαμε το κλειδί, αυτό τοποθετείται στο μηχανισμό που συγκρατεί τα κλειδιά στον τοπικό υπολογιστή. Δημιουργεί μάλιστα ένα αρχείο με το όνομα .keystore (βλέπε Σχήμα 6-49) στο φάκελο user.home. Δηλαδή στο φάκελο:

```
C:\Documents and Settings\uName στα Windows XP

C:\Users\uName στα Windows Vista
```



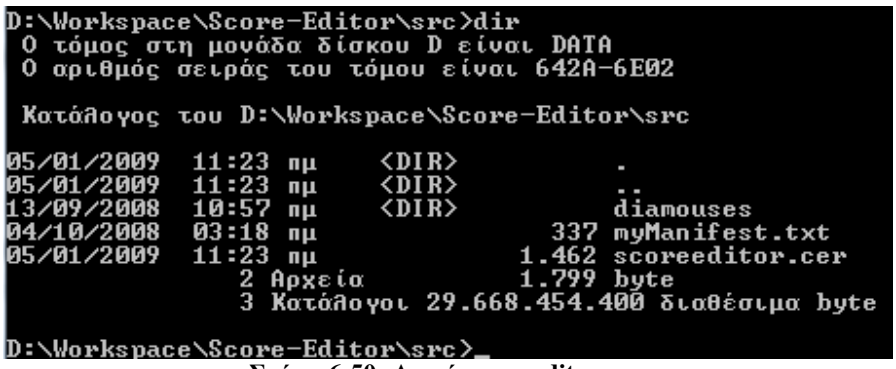
Σχήμα 6-49: Αρχείο .keystore.

Για να τοποθετήσουμε το certificate στο φάκελο στον οποίο πατάμε την εντολή, πρέπει να χρησιμοποιήσουμε την παράμετρο -keystore ορίζοντας το όνομα του αρχείου που επιθυμούμε να χρησιμοποιήσουμε.

Έτσι χρησιμοποιώντας την εντολή

```
keytool -genkey -alias score_editor -keyalg rsa -validity 750 -
                    keystore scoreeditor.cer
```

Θα δημιουργηθεί το αρχείο scoreeditor.cer στον τρέχον φάκελο όπως φαίνεται στο Σχήμα 6-50.



Σχήμα 6-50: Αρχείο scoreeditor.cer.

Το επόμενο βήμα είναι να κάνουμε sign το JAR αρχείο μας με το κλειδί που μόλις δημιουργήσαμε. Αφού το κάνουμε και αυτό προσπαθούμε να τρέξουμε το applet και να σώσουμε ένα MIDI αρχείο στον τοπικό δίσκο του χρήστη. Παρατηρούμε όμως το εξής μήνυμα λάθους.

```
Going to save score into file : C:\Users\Desktop\test.midi
-------------------- Writing MIDI File -------------------------------
Converting to SMF data structure...
    Part 0 'Untitled Part' to SMF Track on Ch. 0:  Phrase 0:..
Exception in thread "AWT-EventQueue-2"
java.security.AccessControlException: access denied
(java.io.FilePermission C:\Users\Desktop\test.midi write)
      at java.security.AccessControlContext.checkPermission(Unknown
      Source)
      at java.security.AccessController.checkPermission(Unknown Source)
      at java.lang.SecurityManager.checkPermission(Unknown Source)
```

Ο Security Manager της Java Virtual Machine αρνείται να σώσει το αρχείο test.midi παρόλο που το Applet μας είναι signed και ο χρήστης δέχεται την παροχή αυξημένων δικαιωμάτων. Μετά από αρκετό trial & error καταλάβαμε ότι πέρα από το Score_Editor.jar πρέπει να κάνουμε sign και τη βοηθητική βιβλιοθήκη jmusic.jar

Έτσι συνολικά οι εντολές που πρέπει να πληκτρολογήσουμε για να πετύχουμε τον αρχικό στόχο, την παροχή δικαιωμάτων εγγραφής στον τοπικό δίσκο είναι:

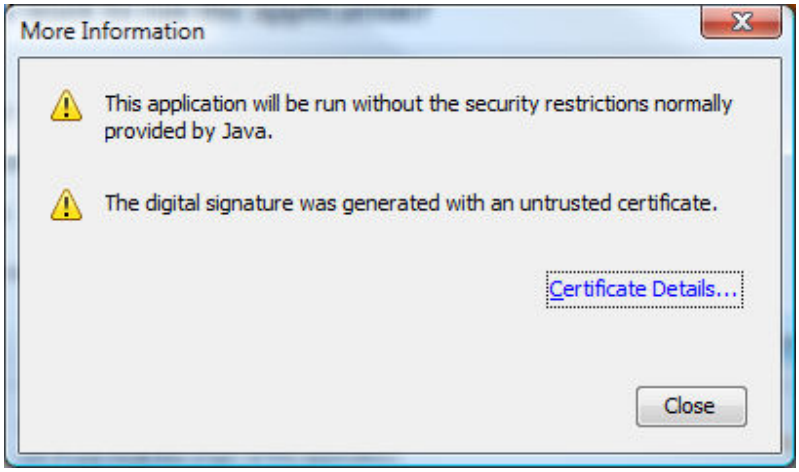
1. **keytool –genkey –alias score_editor**
2. **keytool –export –alias score_edit –rfc –file score_editor_sig.x509**
3. **jarsigner ScoreEditor.jar score_editor**
4. **jarsigner jmusic.jar score_editor**

Κατόπιν και χρησιμοποιώντας τα electronically signed applets, όταν ο χρήστης επισκέπτεται τη σελίδα του εμφανίζεται το μήνυμα που παρουσιάζεται στο Σχήμα 6-51, και πρέπει να επιλέξει ‘Run’, διαφορετικά δεν θα δώσει τα απαραίτητα δικαιώματα στο Applet.

Σχήμα 6-51: Μήνυμα στο χρήστη που αφορά τα δικαιώματα του Applet.

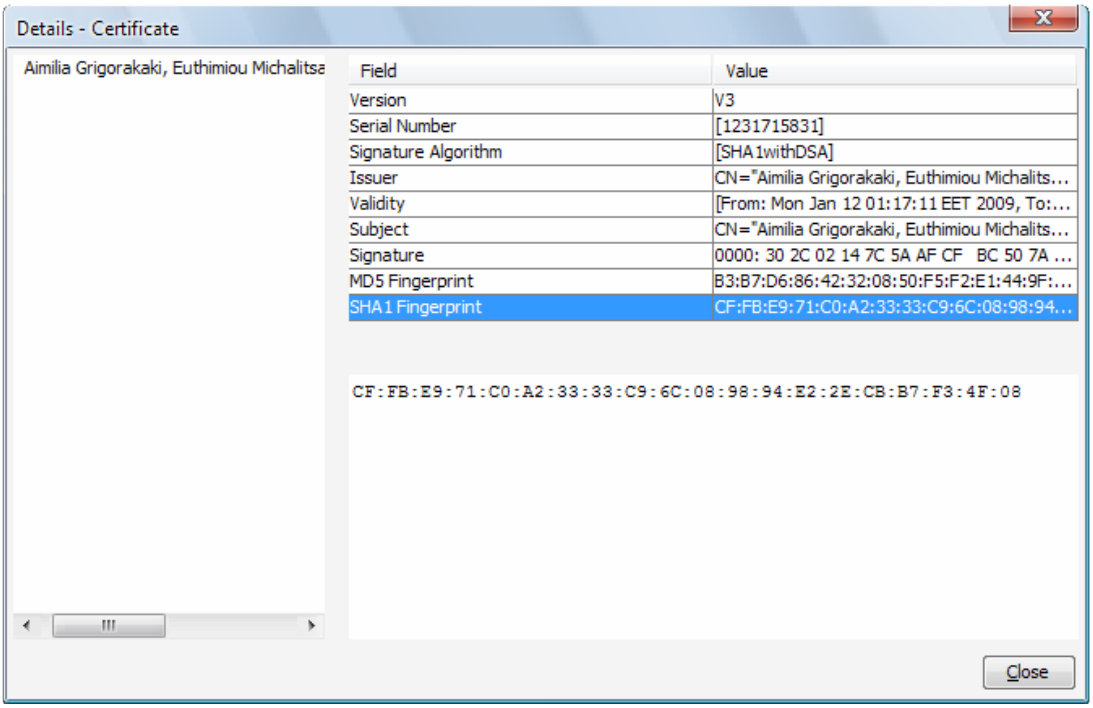
Στο κάτω δεξιά μέρος της παραπάνω εικόνας υπάρχει ένα link με την ονομασία More Information (βλέπε Σχήμα 6-52). Ο χρήστης επιλέγοντας το link αυτό μπορεί να δει πληροφορίες σχετικές με το πιστοποιητικό ασφαλείας. Μπορεί να δει δηλαδή τα στοιχεία που συμπληρώσαμε όταν δημιουργούσαμε το πιστοποιητικό αυτό. Πρώτα βέβαια εμφανίζεται το παρακάτω μήνυμα που προειδοποιεί τον χρήστη για 2 πράγματα: α) ότι αν δεχθεί το πιστοποιητικό θα δώσει αυξημένα δικαιώματα σε remote applet και β) ότι το συγκεκριμένο πιστοποιητικό είναι untrusted.

Θα μπορούσαμε να δημιουργούσαμε ένα trusted πιστοποιητικό, αλλά αυτό προϋποθέτει την πληρωμή μιας συνδρομής σε έναν από τους διεθνής αναγνωρισμένους οργανισμούς που έχουν αναλάβει την παροχή trusted πιστοποιητικών με ένα σχετικά μικρό κόστος. Στην περίπτωση αυτή θα στέλναμε το public key στον οργανισμό αυτό (π.χ. VeriSign) και θα μας το πιστοποιούσε αφού εξακρίβωνε τα πραγματικά μας στοιχεία. Το private key όμως θα το κρατούσαμε εμείς κάνοντάς μας τους μοναδικούς που θα μπορούσαν να κάνουν sign applets με το συγκεκριμένο κλειδί - ηλεκτρονική υπογραφή μας. Κάτι τέτοιο όμως δεν είναι απαραίτητο και δεν έγινε στα πλαίσια της πτυχιακής αυτής εργασίας.



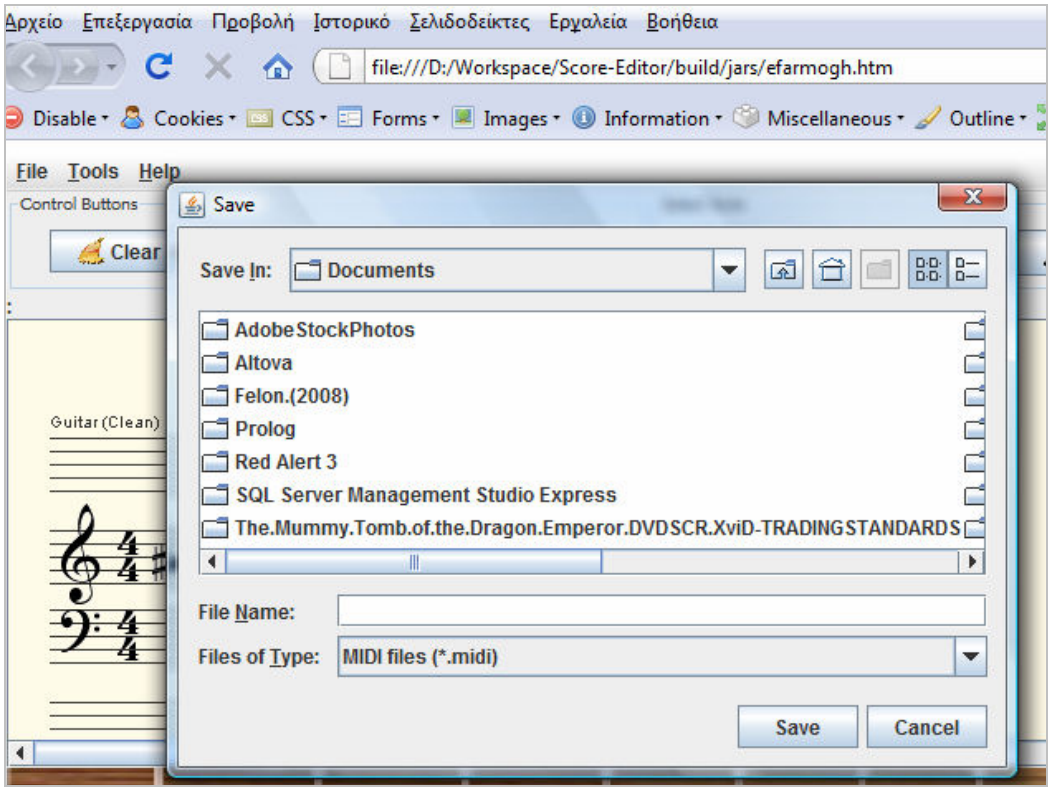
Σχήμα 6-52: Παράθυρο 'More Information'.

Τέλος πατώντας 'Certificate Details…' μπορούμε να δούμε τα στοιχεία που συνοδεύουν το πιστοποιητικό ασφαλείας, όωπς φαίνεται στο Σχήμα 6-53.



Σχήμα 6-53: Στοιχεία που συνοδεύουν το πιστοποιητικό ασφαλείας.

Εν τέλει ο χρήστης μπορεί να αποθηκεύσει αρχεία στον τοπικό υπολογιστή του χρήστη και να κάνει χρήση του JFileChooser component που φαίνεται στο Σχήμα 6-54.



Σχήμα 6-54: Απεικόνιση Save Dialog για αποθήκευση παρτιτούρας.

6.11 Δημιουργία Ιστοσελίδας

Το Joomla [33] είναι ένα πλήρες σύστημα διαχείρισης περιεχομένου ανοιχτού κώδικα. Είναι εφαρμογή η οποία μπορεί να καλύψει τις ανάγκες μιας προσωπικής ιστοσελίδας, αλλά και ενός ολόκληρου δικτυακού τόπου. Είναι προσαρμόσιμο σε περιβάλλοντα επιχειρηματικής κλίμακας όπως τα intranets μεγάλων επιχειρήσεων ή οργανισμών, και οι δυνατότητες επέκτασής του είναι πρακτικά απεριόριστες.

Το Joomla εγκαθίσταται σε έναν κεντρικό υπολογιστή, τον web server. Ο χρήστης - διαχειριστής , έχει πρόσβαση στο περιβάλλον διαχείρισης του Joomla (βλέπε Σχήμα 6-55) μέσω ενός browser, όπως είναι ο Internet Explorer ή ο Firefox, και μπορεί να προσθέσει οποιοδήποτε κείμενο ή γραφικό ώστε να δημιουργήσει την ιστοσελίδα του.

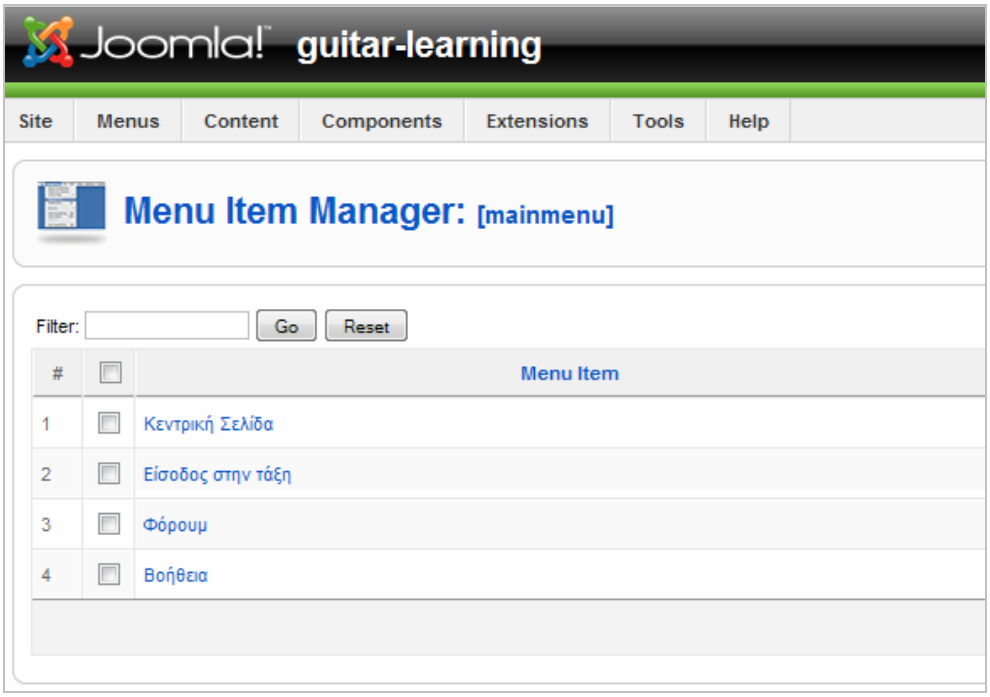
Σχήμα 6-55: Περιβάλλον διαχείρισης Joomla.

Το Joomla χρησιμοποιεί μια ισχυρή templating engine που μας δίνει την δυνατότητα να χρησιμοποιήσουμε το template που επιθυμούμε. Έτσι, ψάχνοντας στον παγκόσμιο ιστό βρήκαμε το template [34] που πληρούσε τις δικές μας προϋποθέσεις από αισθητικής άποψης, και το εγκαταστήσαμε στο Joomla. Στη συνέχεια εργαστήκαμε στο υπάρχον template αλλάζοντας από το αρχείο css [35] παραμέτρους που αφορούν την εμφάνιση.

Η τελική μορφή που πήρε το template της σελίδας μας απεικονίζεται στο Σχήμα 6-56.

Σχήμα 6-56: Template ιστοσελίδας.

Αφού ολοκληρώσαμε την επεξεργασία του template και γενικά της εμφάνισης που θα έχει η ιστοσελίδα μας, προχωρήσαμε στο επόμενο βήμα που είναι η προσθήκη των υποσελίδων που θα αποτελούν την δομή της σελίδας. Αρχικά μέσα από το control panel του Joomla ορίσαμε την εμφάνιση του κεντρικού μενού και έπειτα προσθέσαμε τα menu items που θα απαρτίζουν το κεντρικό μενού της σελίδας μας, όπως φαίνεται στο Σχήμα 6-57.

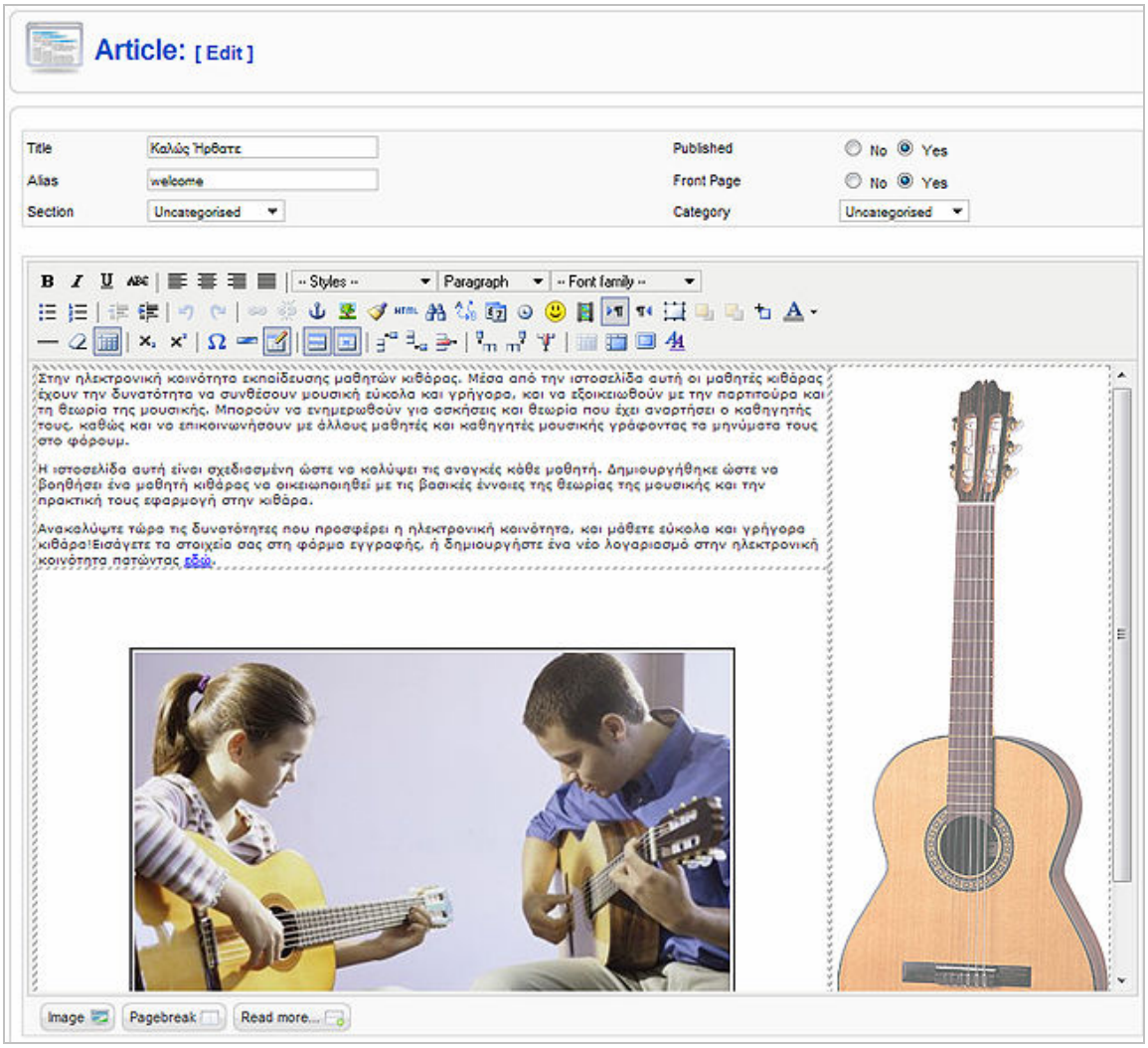


Σχήμα 6-57: Κεντρικό μενού της σελίδας.

Αφού δημιουργήσαμε το κεντρικό μενού της σελίδας ξεκινήσαμε να προσθέτουμε περιεχόμενο σε κάθε υποσελίδα.

Κεντρική Σελίδα

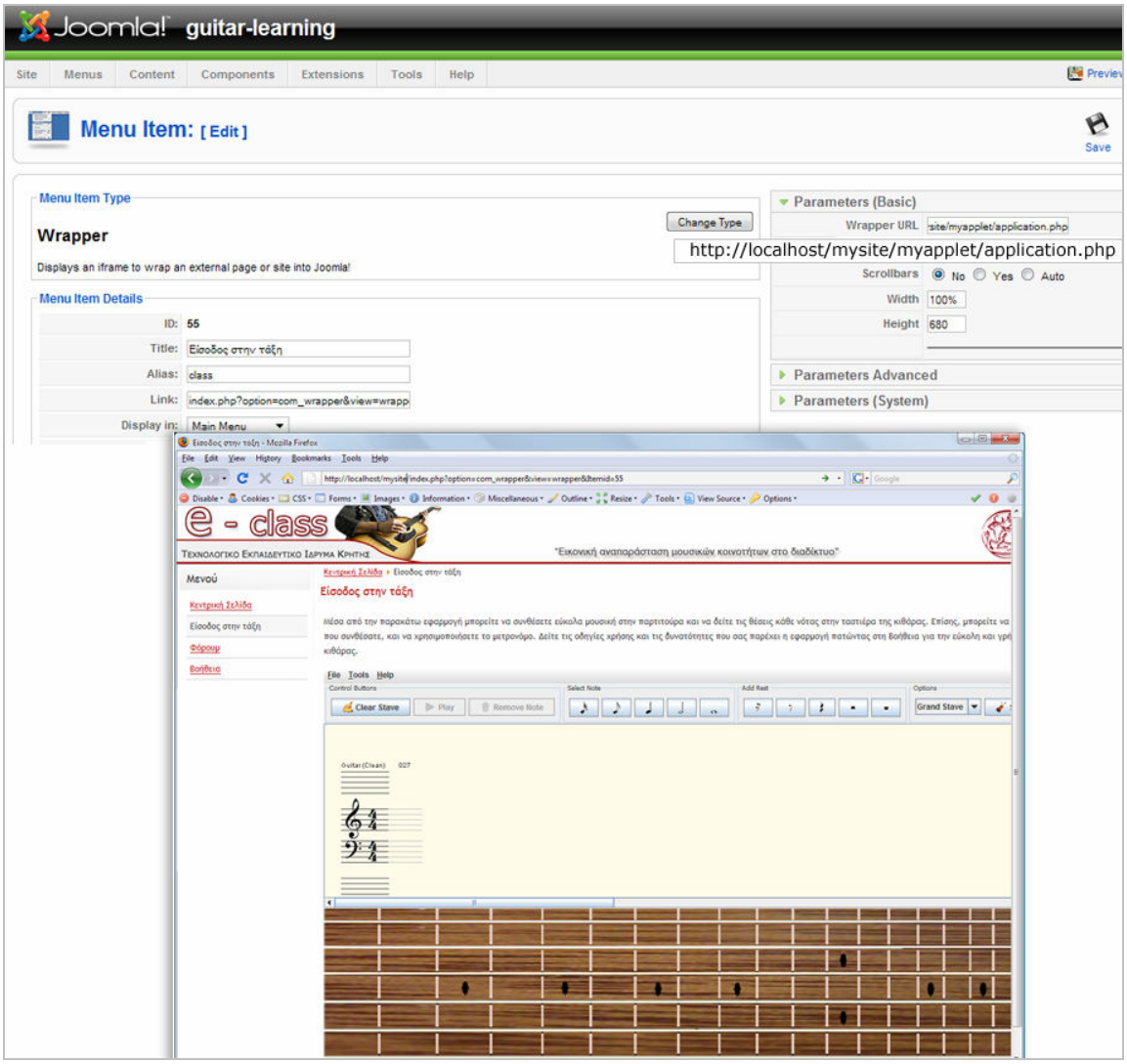
Πηγαίνοντας στο μενού Content => Article Manager του Joomla μπορούμε να διαχειριστούμε το περιεχόμενο κάθε υποσελίδας. Έτσι, η κεντρική σελίδα διαμορφώθηκε όπως φαίνεται στο Σχήμα 6-58.



Σχήμα 6-58: Κεντρική σελίδα.

Είσοδος Στην Τάξη

Σε αυτή την σελίδα εργαστήκαμε διαφοτερικά διότι έπρεπε να ενσωματώσουμε το applet μέσα στο οποίο θα εμφανίζεται η εφαρμογή μας. Αφού γράψαμε τον κώδικα που ήταν απαραίτητος για την εμφάνιση του applet σε μια html σελίδα προσθέσαμε ένα εισαγωγικό κείμενο πριν την εφαρμογή, και ορίσαμε στο Joomla το Menu Item «Είσοδος στην τάξη» ως Wrapper. Με αυτό τον τρόπο το Joomla δημιουργεί ένα εσωτερικό πλαίσιο (iframe) όπου μπορούμε να ενσωματώσουμε μία εξωτερική σελίδα στη σελίδα μας. Έτσι εισάγουμε το url της σελίδας (http://localhost/mysite/myapplet/application.php) στην οποία εμφανίζουμε το applet και παίρνουμε το αποτέλεσμα που απεικονίζεται στο Σχήμα 6-59.

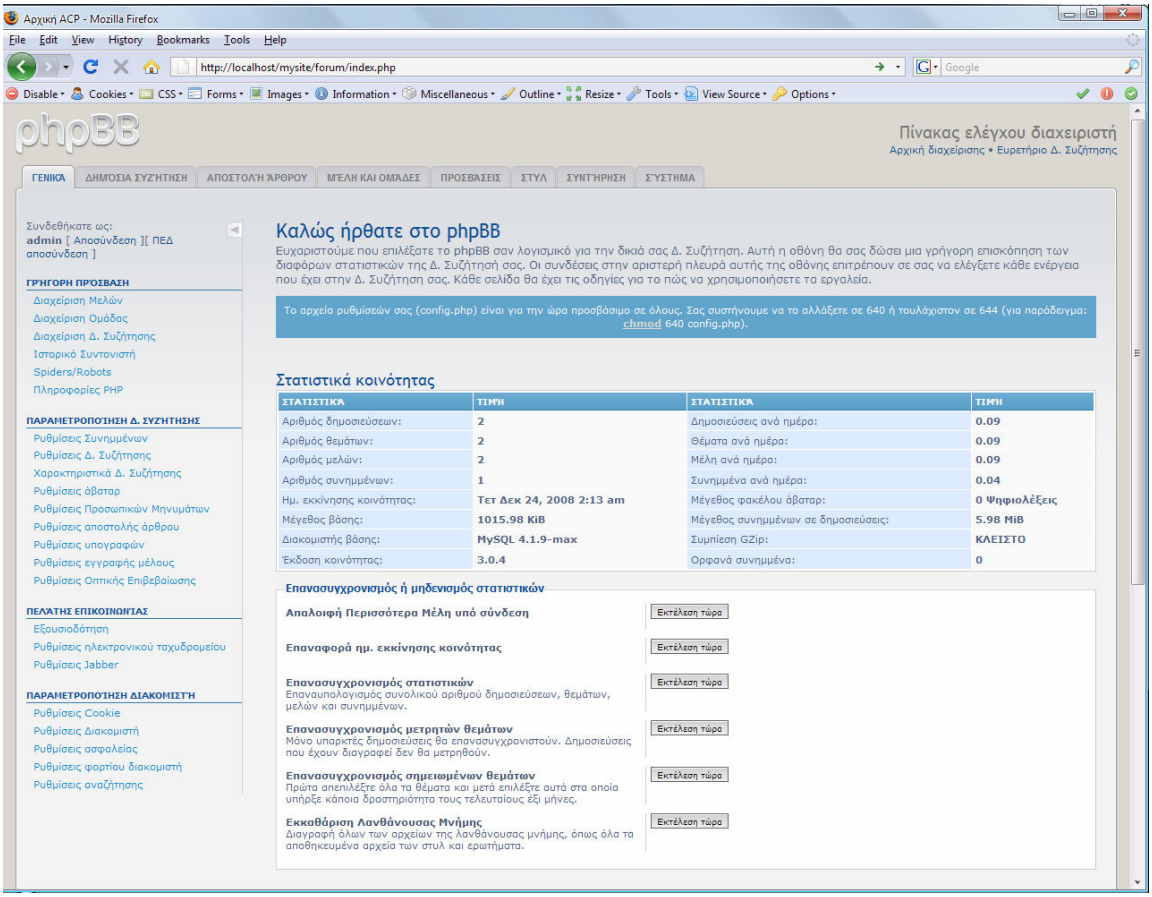


Σχήμα 6-59: Σελίδα 'Είσοδος στην τάξη'.

Φόρουμ

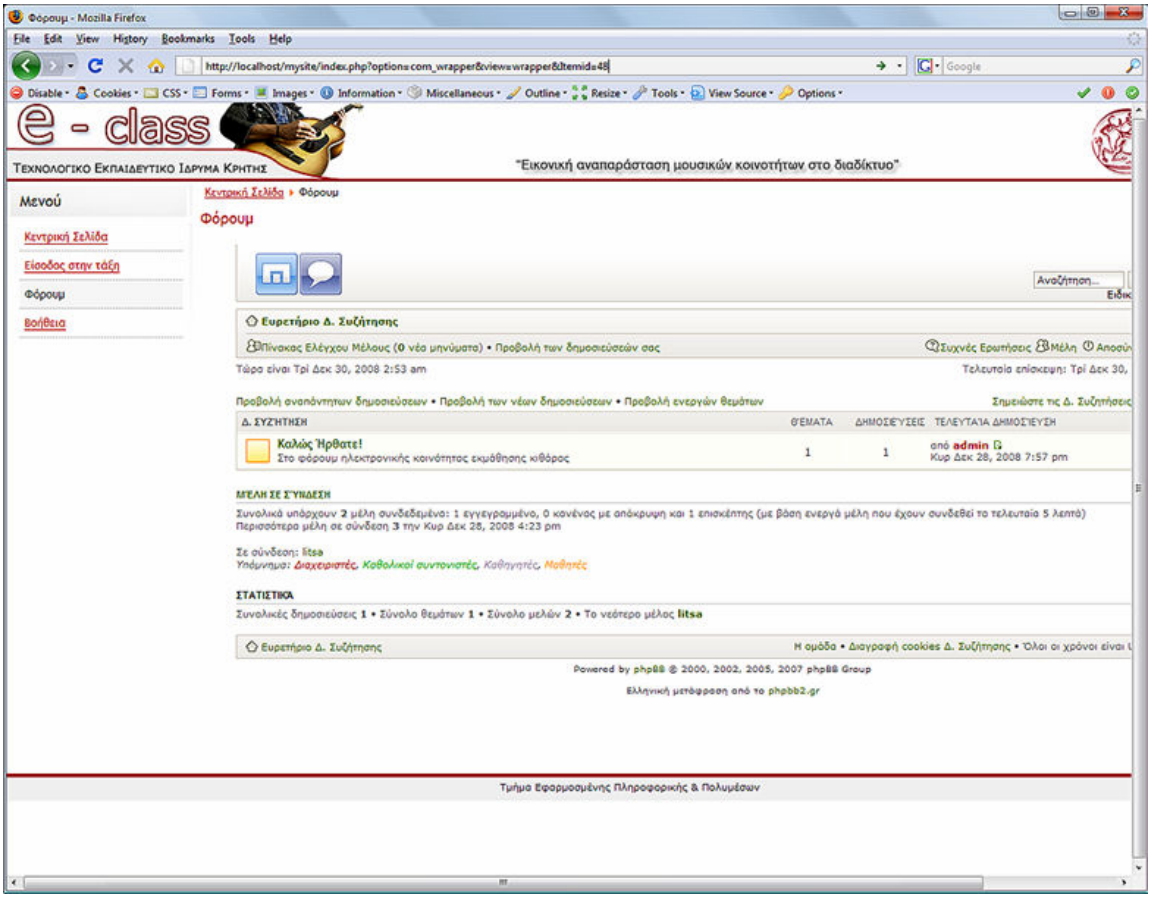
Για το φόρουμ εργαστήκαμε ως εξής:

Αρχικά κατεβάσαμε και εγκαταστήσαμε στην ίδια βάση με το Joomla, το phpBB3 ("Olympus") forum [36] που είναι ένα forum ανοιχτού κώδικα. Μετά την εγκατάσταση του phpBB3 στον υπολογιστή βλέπουμε τον πίνακα ελέγχου του διαχειριστή (ACP) του φόρουμ (βλέπε Σχήμα 6-60).



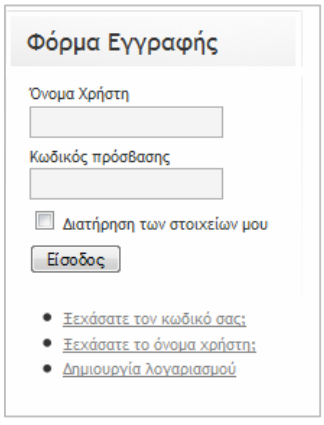
Σχήμα 6-60: Πίνακας ελέγχου διαχειριστή φόρουμ.

Έπειτα ψάξαμε στον παγκόσμιο ιστό το στυλ του φόρουμ που ταιριάζει στη σελίδα μας και αφού το εγκαταστήσαμε το ορίσαμε ως default για το φόρουμ. Η ενσωμάτωση του φόρουμ στη σελίδα έγινε με παρόμοιο τρόπο όπως και στην σελίδα «Είσοδος Στην Τάξη». Δηλαδή ορίσαμε το Menu Item «Φόρουμ» ως Wrapper και βάλαμε το url του φόρουμ phpBB3 που στήσαμε. Έτσι η σελίδα «Φόρουμ» διαμορφώθηκε όπως φαίνεται στο Σχήμα 6-61.



Σχήμα 6-61: Σελίδα 'Φόρουμ'.

Τέλος, αποφασίσαμε η σελίδα να υποστηρίζει login χρηστών. Κάνοντας enabled το login module του Joomla, εμφανίστηκε στην σελίδα η φόρμα εγγραφής που φαίνεται στο Σχήμα 6-62.



Σχήμα 6-62: Φόρμα εγγραφής χρηστών.

Αυτό είχε ως αποτέλεσμα κάθε φορά που ο χρήστης έκανε login στη σελίδα να αναγκάζεται να κάνει login και στο φόρουμ. Το πρόβλημα λύθηκε εγκαθιστώντας μια γέφυρα [37] μεταξύ του Joomla και του phpBB3 forum. Με αυτό τον τρόπο οι χρήστες κάνουν μια φορά login στη σελίδα και φαίνονται συνδεδεμένοι και στο φόρουμ.

Το phpBB3 forum μας δίνει τη δυνατότητα μέσω του πίνακα ελέγχου να δημιουργήσουμε νέες δημόσιες συζητήσεις, να ορίσουμε τις προσβάσεις που θα έχουν οι ομάδες σε αυτές, και τους ρόλους των ομάδων μελών. Επίσης μπορούμε να δημιουργήσουμε νέες ομάδες μελών και να ορίσουμε επιπλέον δυνατότητες σε αυτές. Συγκεκριμένα δημιουργήσαμε δύο νέες ομάδες, τις *Καθηγητές* και *Μαθητές*, και τις εξής δημόσιες συζητήσεις:

- Καλώς Ήρθατε
- Θεωρία Ασκήσεις
- Απορίες για τη θεωρία και τις ασκήσεις
- Γενικά

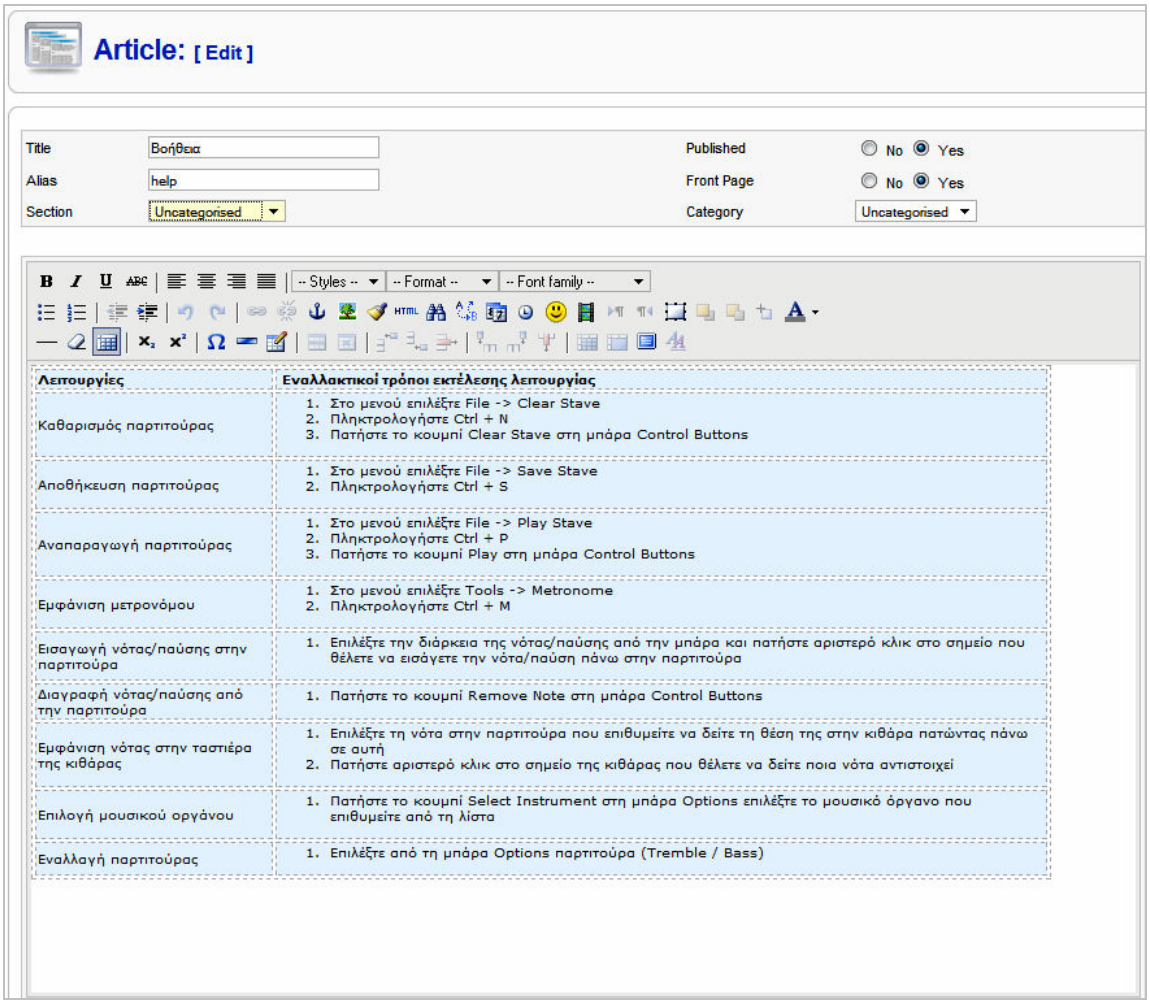
Σε κάθε μία από αυτές τις δημόσιες συζητήσεις ορίσαμε τα δικαιώματα πρόσβασης συνοψίζονται στον Πίνακα 6-2.

Δημόσια **Σ**υζήτηση	**Δ**ικαιώματα **Π**ρόσβασης
Θεωρία Ασκήσεις	*μόνο οι καθηγητές έχουν τη δυνατότητα να επισυνάπτουν αρχεία και μόνο οι μαθητές μπορούν να μεταφορτώσουν τα αρχεία αυτά. Επίσης μόνο οι καθηγητές μπορούν να δημιουργήσουν νέα θέματα σε αυτή τη δημόσια συζήτηση*
Απορίες για τη θεωρία και τις ασκήσεις	*οι μαθητές μπορούν να συζητήσουν τις απορίες που έχουν σχετικά με τη θεωρία και τις ασκήσεις που έχουν επισυνάψει οι καθηγητές για αυτούς. Επίσης οι μαθητές μπορούν να επισυνάψουν αρχεία ώστε να διευκολύνουν τους καθηγητές και να λυθούν ευκολότερα οι απορίες τους*
Γενικά	*μπορούν όλα τα μέλη να συζητήσουν γενικά για ότι αφορά την κιθάρα. Οι επισκέπτες του φόρουμ μπορούν μόνο να διαβάσουν τα θέματα που έχουν δημοσιευτεί σε αυτή τη δημόσια συζήτηση. Δεν έχουν όμως δικαίωμα συμμετοχής στη συζήτηση με την δημιουργία νέου θέματος*

Πίνακας 6-2: Δικαιώματα πρόσβασης στις Δημόσιες Συζητήσεις.

Βοήθεια

Στη «Βοήθεια» εργαστήκαμε όπως και στην «Κεντρική Σελίδα» και έτσι η σελίδα διομορφώθηκε όπως φαίνεται στο Σχήμα 6-63.



Σχήμα 6-63: Σελίδα 'Βοήθεια'.

Επίπεδο πρόσβασης στις σελίδες

Το επίπεδο πρόσβασης των χρηστών στις σελίδες παρουσιάζεται στον Πίνακα 6-3.

Σελίδα	Επίπεδο Πρόσβασης Χρηστών
Κεντρική Σελίδα	*Δημόσιο / Public*
Είσοδος στην Τάξη	*Μέλη / Registered*
Φόρουμ	*Δημόσιο / Public*
Βοήθεια	*Μέλη / Registered*

Πίνακας 6-3: Επίπεδο πρόσβασης χρηστών στις σελίδες.

Όπως προκύπτει από τον πίνακα όλοι οι επισκέπτες της σελίδας έχουν πρόσβαση στην κεντρική σελίδα και στο φόρουμ, ενώ οι εγγεγραμένοι χρήστες έχουν επιπλέον πρόσβαση στην τάξη και στην βοήθεια.

7. Σύνοψη και Συμπεράσματα

Με την εκπόνηση της πτυχιακής αυτής εργασίας, αποκτήσαμε γνώσεις σχετικά με τον αντικειμενοστραφή προγραμματισμό χρησιμοποιώντας την Java. Ανακαλύψαμε τις τεράστιες δυνατότητες που παρέχει ο αντικειμενοστραφής προγραμματισμός, υλοποιώντας απλές μεθόδους και κλάσεις, και δημιουργώντας αντικείμενα.

Επίσης, με την χρήση του εργαλείου Joomla διαπιστώσαμε την αξία ενός συστήματος διαχείρισης περιεχομένου ιστοσελίδων. Χρησιμοποιώντας modules και components του Joomla δημιουργήσαμε την ιστοσελίδα που πληρούσε τις δικές μας απαιτήσεις και προδιαγραφές, με απλές διαδικασίες.

Η πτυχιακή αυτή εργασία μας εφοδίασε με γνώσεις τόσο στον αντικειμενοστραφή προγραμματισμό, όσο και στον προγραμματισμό ιστοσελίδων, τις οποίες θα αξιοποιήσουμε στην μελλοντική εργασία μας.

Οι δυνατότητες επέκτασης και βελτίωσης της πτυχιακής εργασίας είναι πολλές. Μπορεί να αποτελέσει τη βάση ώστε να προστεθούν και άλλες εικονικές αναπαραστάσεις κιθάρας (π.χ. ηλεκτρική κιθάρα). Έχουμε υλοποιήσει έτσι την εφαρμογή ώστε να μπορεί να παραμετροποιηθεί εύκολα το αντικείμενο GuitarNeck και να καθιστά απλή μια τέτοια βελτίωση. Επίσης, θα μπορούσαν να προστεθούν και άλλα μουσικά όργανα ώστε ο χρήστης όταν επιλέγει ένα μουσικό όργανο να βλέπει και την αντίστοιχη απεικόνισή του. Ακόμα, θα μπορούσε να βελτιωθεί το κουμπί play score, ώστε όταν ο χρήστης το πατάει, για κάθε νότα που θα ακούγεται να εμφανίζεται η εκάστοτε θέση της στην εικονική αναπαράσταση του μουσικού οργάνου.

Γενικά οι επεκτάσεις και οι βελτιώσεις της πτυχιακής εργασίας περιορίζονται μόνο από την φαντασία του εκάστοτε προγραμματιστή!

8. Βιβλιογραφία

[1] UML Διαγράμματα,
http://el.wikipedia.org/wiki/Γλώσσες_μοντελοποίησης_λογισμικού

[2] Βιβλιοθήκη ABC4J,
http://code.google.com/p/abc4j/

[3] ABC notation,
http://en.wikipedia.org/wiki/Abc_notation

[4] Using abc notation abc4j,
http://code.google.com/p/abc4j/wiki/JScore_Using_abc_notation

[5] Αρχεία MIDI,
http://el.wikipedia.org/wiki/MIDI

[6] ImproVisor,
http://www.cs.hmc.edu/~keller/jazz/improvisor/

[7] JFugue,
http://www.jfugue.org/

[8] Java MIDI Kit,
http://www.mcnabb.com/software/fantasia/index.html

[9] jMusic,
http://jmusic.ci.qut.edu.au/

[10] jFrets,
https://jfrets.dev.java.net/

[11] ABC BNF,
http://www.norbeck.nu/abc/abcbnf.htm

[12] The Complete Guide to JFugue,
http://www.jfugue.org/book.html

[13] Λίστα με βιβλιοθήκες και πακέτα λογισμικού στη μουσική τεχνολογία,
http://www.softsynth.com/links/java_music.html

[14] Java Sound API,
http://java.sun.com/products/java-media/sound/

[15] Apple QuickTime,
http://www.apple.com/quicktime/

[16] MidiShare,
http://midishare.sourceforge.net/

[17] CodeSounding,
http://www.codesounding.org/

[18] Red Wine Music,
http://www.redwinemusic.com/

[19] JM-Etude,
http://jmetude.dihardja.de/

[20] CVS,
http://en.wikipedia.org/wiki/Concurrent_Versions_System

[21] Subversion,
http://subversion.tigris.org/

[22] NetBeans,
http://www.netbeans.org/

[23] Rational Rose,
http://www-01.ibm.com/software/rational/

[24] Κληρονομικότητα,
http://www.cs.teilar.gr/gkakaron/java/Day2/8.html

[25] JAR File Specifications,
http://java.sun.com/j2se/1.4.2/docs/guide/jar/jar.html

[26] Ant tool,
http://ant.apache.org/

[27] Singleton,
http://en.wikipedia.org/wiki/Singleton_pattern

[28] Design Patterns,
http://en.wikipedia.org/wiki/Design_pattern_(computer_science)

[29]Guitar note positioning,
http://www.ocmusic.com/trebleclef.htm

[30] Saving MIDI files in JMusic,
http://jmusic.ci.qut.edu.au/jmtutorial/jmDOSTute.html

[31] FileChoosers in Java,
http://java.sun.com/docs/books/tutorial/uiswing/components/filechooser.html

[32] Θεωρία μουσικής,
http://homepages.pathfinder.gr/papakrasas/diafora.htm

[33] Joomla,
http://joomla.org

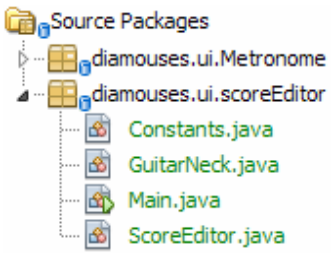
[34] Template,
http://www.themza.com/joomla1.5/computer-society-template.html

[35] Cascading Style Sheets,
http://www.w3schools.com/css

[36] PhpBB3,
http://www.phpbb.com

[37] Γέφυρα Joomla – phpBB3,
http://www.rocketwerx.com/forum/viewtopic.php?f=22&t=368

Παράρτημα 1




```
package diamouses.ui.scoreEditor;

import javax.swing.ImageIcon;

public final class Constants
{

    public ImageIcon a;
    public  ImageIcon asharp;
    public  ImageIcon b ;
    public  ImageIcon c;
    public  ImageIcon csharp;
    public  ImageIcon d;
    public  ImageIcon dsharp;
    public  ImageIcon e;
    public  ImageIcon f;
    public  ImageIcon fsharp;
    public  ImageIcon g;
    public  ImageIcon gsharp;


    public Constants()
    {
        a = new ImageIcon(getClass().getResource("images/note_images/a.png"));
        asharp = new
ImageIcon(getClass().getResource("images/note_images/asharp.png"));
        b = new ImageIcon(getClass().getResource("images/note_images/b.png"));
        c = new ImageIcon(getClass().getResource("images/note_images/c.png"));
        csharp = new
ImageIcon(getClass().getResource("images/note_images/csharp.png"));
        d = new ImageIcon(getClass().getResource("images/note_images/d.png"));
        dsharp = new
ImageIcon(getClass().getResource("images/note_images/dsharp.png"));
        e = new ImageIcon(getClass().getResource("images/note_images/e.png"));
        f = new ImageIcon(getClass().getResource("images/note_images/f.png"));
        fsharp = new
ImageIcon(getClass().getResource("images/note_images/fsharp.png"));
        g = new ImageIcon(getClass().getResource("images/note_images/g.png"));
        gsharp = new
ImageIcon(getClass().getResource("images/note_images/gsharp.png"));
        e.setDescription("e");
        f.setDescription("f");
        fsharp.setDescription("fsharp");
        g.setDescription("g");
        gsharp.setDescription("gsharp");
```

```
        a.setDescription("a");
        asharp.setDescription("asharp");
        b.setDescription("b");
        c.setDescription("c");
        csharp.setDescription("csharp");
        d.setDescription("d");
        dsharp.setDescription("dsharp");
    }

}
```

==========

==========

```
package diamouses.ui.scoreEditor;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.image.BufferedImage;
import java.net.URL;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JToggleButton;

import diamouses.ui.scoreEditor.staves.DStave;

public class GuitarNeck extends JPanel {

      /**
       * Global variables for the application
       */
      private int height = 28;
      private int screen_width;
      private JToggleButton [][] fret_buttons = new JToggleButton
      [6][22];
      private JPanel [] string_panels;
      private JPanel fret_panel;
      ImageIcon ii;
      Color color_1st = new Color(241,255,255);
      Color color_2nd = new Color(255,255,240);
      Color color_3rd = new Color(255,255,230);
      Color color_4th = new Color(255,255,225);
      Color color_5th = new Color(255,255,220);
      Color color_6th = new Color(255,255,210);

      private static GuitarNeck reference;
      /**
       * Constructor.
       * Initializes 6 JPanels, one for each guitar string
       */
      private GuitarNeck(int screen_width) {
```

```
        this.screen_width = screen_width;
        // 0. Load background image
        String imgLocation2 = "images/GuitarNeck.png";
        URL imageURL = GuitarNeck.class.getResource(imgLocation2);
        ii = new ImageIcon(imageURL, "altText");


        // 1. Create six JPanels
        fret_panel = getStringPanel();

        // 2. Make them Not-Opaque (Make them TRANSPARENT)
        fret_panel.setOpaque(false);

        add(fret_panel);

        setSizes();
}

public static GuitarNeck getInstance(int mm) {
        if (reference==null) {
                reference = new GuitarNeck(mm); // mm = 1160px
        }
        return reference;
}

public static GuitarNeck getInstance() {
        return reference;
}

/**
 * Method returns a JPanel with JButtons for each note of the
   Guitar
 * @param string First..Sixth string of the guitar
 * @return a JPanel with appropriate JButtons
 */
private JPanel getStringPanel() {
        JPanel main_panel = new JPanel();
        ImageIcon [] string_images;
        // Add each of the JPanel in a stack from top to bottom
        main_panel .setLayout(new GridLayout(0,1,0,0));
        string_panels = new JPanel [6];

        for (int string = 0; string <= 5; string++) {
              string_panels[string] = new JPanel();
              string_panels[string].setOpaque(false);
              string_images = getIconsForString(string);
              for (int i = 0; i <= 21; i++) {
               fret_buttons[string][i] = new JToggleButton();
               fret_buttons[string][i].setContentAreaFilled(false);
               fret_buttons[string][i].setBorderPainted(false);
               fret_buttons[string][i].setOpaque(false);
                fret_buttons[string][i].setIcon(new ImageIcon());
                fret_buttons[string][i].setSelectedIcon(string_images
                [i]);
                string_panels[string].add(fret_buttons[string][i]);
              }
```

```
                main_panel.add(string_panels[string]);
        }
        return main_panel;
}

/**
 * This method sets the sizes of the JToggleButtons
 */
private void setSizes () {
        int width;
        for (int i = 0; i <= 21; i++) {
        switch (i) {
                case 0: width = 83; break;
                case 1: width = 89; break;
                case 2: width = 82; break;
                case 3: width = 77; break;
                case 4: width = 75; break;
                case 5: width = 66; break;
                case 6: width = 64; break;
                case 7: width = 59; break;
                case 8: width = 55; break;
                case 9: width = 54; break;
                case 10: width = 49; break;
                case 11: width = 46; break;
                case 12: width = 43; break;
                case 13: width = 40; break;
                case 14: width = 38; break;
                case 15: width = 35; break;
                case 16: width = 32; break;
                case 17: width = 30; break;
                case 18: width = 28; break;
                case 19: width = 26; break;
                case 20: width = 22; break;
                default: width = 12; break;
        }
        width+=2;
        width = screen_width*width/1280;
        //string_1.setPreferredSize(new Dimension(width,height));

        fret_buttons[0][i].setPreferredSize(new Dimension(width,
        height));
        fret_buttons[1][i].setPreferredSize(new Dimension(width,
        height));
        fret_buttons[2][i].setPreferredSize(new Dimension(width,
        height));
        fret_buttons[3][i].setPreferredSize(new Dimension(width,
        height));
        fret_buttons[4][i].setPreferredSize(new Dimension(width,
        height));
        fret_buttons[5][i].setPreferredSize(new Dimension(width,
        height));
        }

}


/**
```

```
 * Method Paint the Guitar Neck on the Background of this JPanel.
 */
public void paintComponent(Graphics g) {

  g.drawImage(ii.getImage(),0,0,this.getSize().width,this.getSize()
  .height, this);
}

/**
 * Switch the color of the string
 * @param string values 1 to 6
 */
private void switchColor(int string) {
      Color new_color;

      switch (string) {
            case 1: new_color = color_1st; break;
            case 2: new_color = color_2nd; break;
            case 3: new_color = color_3rd; break;
            case 4: new_color = color_4th; break;
            case 5: new_color = color_5th; break;
            case 6: new_color = color_6th; break;
            default: new_color = Color.black;
      }
    BufferedImage bi = DStave.toBufferedImage(ii.getImage());
     int w = bi.getWidth();
     int h = bi.getHeight();
     int pixel;
     BufferedImage biOut = new BufferedImage(w, h,
     BufferedImage.TYPE_INT_ARGB_PRE);

    for (int x = 0; x < w; x++) {
         for (int y = 0; y < h; y++) {
              pixel = bi.getRGB(x, y);

              if (pixel == (new_color).getRGB()) {
                pixel = (Color.RED).getRGB();
              }
                  biOut.setRGB(x, y, pixel);
          }
    }

    ii = new ImageIcon(DStave.toImage(biOut));

    repaint();
}
/**
 * Method getIconsForString returns an Array [] of ImageIcons
 * with the images of the appropriate Notes for each string of
 * the Guitar.
 *
* @param string An integer from 1 to 6 representing the
First..Sixth string
*        of a Guitar.
* @return an ImageIcon [] array.
*/
private ImageIcon [] getIconsForString( int string ) {
```

```
        Constants c = new Constants();
        string+=1;
        switch (string) {
        //c.e,
        case 1: ImageIcon [] string_1_notes = { c.f, c.fsharp, c.g,
        c.gsharp,c.a,c.asharp,c.b, c.c,c.csharp,c.d,c.dsharp,c.e,c.f,
        null, null, null, null,null,null,null,null,null};
                    return string_1_notes;
        // c.b
        case 2: ImageIcon [] string_2_notes = {c.c,c.csharp,c.d,
        c.dsharp,c.e,c.f, c.fsharp, c.g, c.gsharp,c.a,c.asharp,c.b,
        c.c,null,null, null, null,null,null,null,null,null};
                    return string_2_notes;
        // c.g
        case 3: ImageIcon [] string_3_notes =
        {c.gsharp,c.a,c.asharp,c.b, c.c,c.csharp,c.d, c.dsharp,
        c.e,c.f,c.fsharp, c.g, c.gsharp,null,null,null, null,
        null,null,null,null,null};
                    return string_3_notes;
        // c.d
        case 4: ImageIcon [] string_4_notes = {c.dsharp,
        c.e,c.f,c.fsharp, c.g, c.gsharp,c.a,c.asharp,c.b, c.c,
        c.csharp,c.d, c.dsharp,null,null,null, null,
        null,null,null,null,null};
                    return string_4_notes;
        // c.a
        case 5: ImageIcon [] string_5_notes = {c.asharp,c.b,
        c.c,c.csharp,c.d,c.dsharp,c.e,c.f,c.fsharp, c.g,c.gsharp,c.a,
        c.asharp,null,null,null, null, null,null,null,null,null};
                    return string_5_notes;
        // c.e
        case 6: ImageIcon [] string_6_notes = {c.f,c.fsharp, c.g,
        c.gsharp,c.a,c.asharp,c.b,c.c,c.csharp,c.d,c.dsharp,c.e,c.f,
        null,null, null, null,null,null,null,null,null};
                    return string_6_notes;
        default: System.out.println("Errornous string " + string + "
        in method GuitarFrame.getIconsForString");
                    return null;
        }

}

/**
 * Sets the note with a specific pitch on the virtual guitar
 *
 * @param pitch the pitch value.
 */

public void setNote(int pitch) {
        clearAll();
        int fret_1 = (pitch-77); //5
        int fret_2 = (pitch-72); //4
        int fret_3 = (pitch-68); //5
        int fret_4 = (pitch-63); //5
        int fret_5 = (pitch-58); //5
        int fret_6 = (pitch-53);
```

```
        if (fret_1>=0 & fret_1<=21) {
              fret_buttons[0][fret_1].setSelected(true);
        }
        if (fret_2>=0 & fret_2<=21) {
              fret_buttons[1][fret_2].setSelected(true);
        }
        if (fret_3>=0 & fret_3<=21) {
              fret_buttons[2][fret_3].setSelected(true);
        }
        if (fret_4>=0 & fret_4<=21) {
              fret_buttons[3][fret_4].setSelected(true);
        }
        if (fret_5>=0 & fret_5<=21) {
              fret_buttons[4][fret_5].setSelected(true);
        }
        if (fret_6>=0 & fret_6<=21) {
              fret_buttons[5][fret_6].setSelected(true);
        }
        if (fret_1==-1)
              switchColor(1);
        if (fret_2==-1)
              switchColor(2);
        if (fret_3==-1)
              switchColor(3);
        if (fret_4==-1)
              switchColor(4);
        if (fret_5==-1)
              switchColor(5);
        if (fret_6==-1)
              switchColor(6);

}

public void clearAll() {
      for (int string=0; string <= 5; string++) {
            for (int i=0; i<fret_buttons[0].length; i++) {
                  fret_buttons[string][i].setSelected(false);
            }
      }
      String imgLocation2 = "images/GuitarNeck.png";
      URL imageURL = GuitarNeck.class.getResource(imgLocation2);
      ii = new ImageIcon(imageURL, "altText");
      repaint();
}
/**
 *For testing
 */
/*public static void main(String[] args) {
      JFrame f = new JFrame();
      f.setVisible(true);
      f.setExtendedState(JFrame.MAXIMIZED_HORIZ);

      Toolkit kit = f.getToolkit();
      Dimension screenSize = kit.getScreenSize();
      int screenWidth = screenSize.width;
      System.out.println("screenwidth" + screenWidth);
```

```
            GuitarNeck ff = GuitarNeck.getInstance(screenWidth);
            f.getContentPane().add(ff);
            f.pack();
      }*/
}
```

```
================================================================================
================================================================================
```

```
package diamouses.ui.scoreEditor;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.io.File;

import javax.swing.*;
import javax.swing.filechooser.FileFilter;

import jm.music.data.Phrase;
import jm.music.data.Score;
import jm.util.Write;

import diamouses.ui.Metronome.Metronome;

/**
 *
 */
public class Main extends JApplet implements ActionListener {

      // Global Variables
      private static ScoreEditor score;

      /**
       * Constructor
       */

      public Main() {

            //setJMenuBar(getJMenuBar());

            //-System.out.println("screenwidth" + screenWidth);

      }

      public void init (){

            score = new ScoreEditor();

            //JFrame new_frame = new JFrame("Score Editor");
            //setDefaultCloseOperation(javax.swing.WindowConstants.DISPOS
            E_ON_CLOSE);
```

```
        setJMenuBar(getJMenuBar());
        Toolkit kit = getToolkit();
        Dimension screenSize = kit.getScreenSize();
        int screenWidth = screenSize.width;
        GuitarNeck ff = GuitarNeck.getInstance(screenWidth);
        getContentPane().add(score, BorderLayout.CENTER);
        getContentPane().add(score.getJToolBar(),
        BorderLayout.PAGE_START);
        getContentPane().add(ff, BorderLayout.SOUTH);
        setVisible(true);
        //setExtendedState(JFrame.MAXIMIZED_HORIZ);
        //pack();
}

/**
 * Creates a JMenuBar (File/Tools/Help) with appropriate
 * menu items
 */
public JMenuBar getJMenuBar() {
        // Local Variables
        JMenuBar menuBar;
        JMenu menu;
        JMenuItem menuItem;

        // Create the menu bar.
        menuBar = new JMenuBar();

        // 1. Build the 'File' menu.
        menu = new JMenu("File");
        menu.setMnemonic(KeyEvent.VK_F);
        menuBar.add(menu);

        // 'New Stave' MenuItem
        menuItem = new JMenuItem("Clear Stave", KeyEvent.VK_N);
        menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
        ActionEvent.CTRL_MASK));
        menuItem.setActionCommand("Clear_Stave");
        menuItem.addActionListener(this);
        menu.add(menuItem);


        // 'Save Stave' MenuItem
        menuItem = new JMenuItem("Save Stave"); //, new
        ImageIcon("images/middle.gif"));
        menuItem.setMnemonic(KeyEvent.VK_S);
        menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
        ActionEvent.CTRL_MASK));
        menuItem.setActionCommand("Save_Stave");
        menuItem.addActionListener(this);
        menu.add(menuItem);

        // Separator
        menu.addSeparator();



        // 'Play Stave' MenuItem
```

```
        menuItem = new JMenuItem("Play Stave"); //, new
        ImageIcon("images/middle.gif"));
        menuItem.setMnemonic(KeyEvent.VK_P);
        menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P,
        ActionEvent.CTRL_MASK));
        menuItem.setActionCommand("Play_Stave");
        menuItem.addActionListener(this);
        menu.add(menuItem);

        // Separator
        menu.addSeparator();

        /* 'Exit' MenuItem
        menuItem = new JMenuItem("Exit"); //, new
        ImageIcon("images/middle.gif"));
        menuItem.setMnemonic(KeyEvent.VK_X);
        menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_X,
        ActionEvent.CTRL_MASK));
        menuItem.setActionCommand("Exit");
        menuItem.addActionListener(this);
        menu.add(menuItem);
        */
        // 2. Build 'Tools' menu
        menu = new JMenu("Tools");
        menu.setMnemonic(KeyEvent.VK_T);
        menuBar.add(menu);

        // 'Metronome' MenuItem
        menuItem = new JMenuItem("Metronome", KeyEvent.VK_M);
        menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_M,
        ActionEvent.CTRL_MASK));
        menuItem.setActionCommand("Metronome");
        menuItem.addActionListener(this);
        menu.add(menuItem);

        // 3. Build 'Help' menu
        menu = new JMenu("Help");
        menu.setMnemonic(KeyEvent.VK_H);
        menuBar.add(menu);

        // 'About' MenuItem
        menuItem = new JMenuItem("About", KeyEvent.VK_A);
        menuItem.setActionCommand("About");
        menuItem.addActionListener(this);
        menu.add(menuItem);

        return menuBar;
}

/**
 * Starting place of the application
 * @param args
 */


public static void main(String args[]) {
```

```
        JFrame f = new JFrame("Πτυχιακή ΤΕΙ Κρήτης -  Ευθυμίου
        Μιχαλίτσα, Γρηγορακάκη Αιμιλία");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JApplet ap = new Main();
        ap.init();
        ap.start();
        f.add("Center", ap);
        f.pack();
        f.setVisible(true);

/*java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                  setDefault();
                  new Main().setVisible(true);
            }
      });
}*/

}
/**
* setDefault() , Sets the default Look and Feel. On Microsoft
Windows
* platforms, this specifies the Windows Look & Feel. On Mac OS
platforms,
* this specifies the Mac OS Look & Feel. On Sun platforms, it
specifies the
* CDE/Motif Look & Feel.
 *
* With JDK1.4.2 upwards it sets the 'Luna' look and feel in Windows
XP and
* 'Bluecurve' in GNOME.
*/
public static void setDefault() {
      try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName())
;
      } catch (java.lang.ClassNotFoundException classE) {
            // Handle exception
      } catch (java.lang.InstantiationException insE) {
            // Handle exception
      } catch (java.lang.IllegalAccessException illE) {
            // Handle exception
      } catch (javax.swing.UnsupportedLookAndFeelException unE) {
            // Handle exception
      }
}

public void actionPerformed(ActionEvent event) {
      String action_command = event.getActionCommand();
      if (action_command.equals("Metronome")) {
            Metronome met = new Metronome(new JFrame(), null,
            true);
        met.setVisible(true);
      }
```

```
        else if (action_command.equals("Clear_Stave")) {
              score.stave.setPhrase(new Phrase());
        }
        else if (action_command.equals("Save_Stave")) {
               String filename = File.separator+".midi";
                JFileChooser fc = new JFileChooser(new
                File(filename));
                fc.setFileFilter(new MIDIFileFilter());

            // Show save dialog; this method does not return until
            the dialog is closed
            fc.showSaveDialog(this);
            File selected_file = fc.getSelectedFile();
            String file_path = selected_file.toString();
            if (!file_path.endsWith(".midi")) {
              file_path += ".midi";
            }

              System.out.println("Goint to save score into file : " +
              file_path);
              Score s = new Score(score.getPart());
              Write.midi(s, file_path);
        }
        else if (action_command.equals("Play_Stave")) {

              score.playTune();
        }
        else if (action_command.equals("About")) {
              JOptionPane.showMessageDialog(new JFrame(),
                        "Αυτή η εφαρμογή αναπτύχθηκε από τις
                        Γρηγορακάκη Αιμιλία και Ευθυμίου
                        Μιχαλίτσα,\n"+
                        "για την εκμάθηση κιθάρας\n"+
                        "Ctrl-N --> Νέα παρτιτούρα\n"+
                        "Ctrl-P --> Play tune\n" +
                        "Ctrl-M --> Show Metronome\n"+
                        //"Ctrl-X --> Exit",
                        "About the score-editor",
                        JOptionPane.INFORMATION_MESSAGE);

        }
        /*else if (action_command.equals("Exit")) {
              this.dispose();
              System.exit(0);
        }*/
}

public static void setEnabledRemove(boolean b) {
        score.setEnabledRemove(b);
        // TODO Auto-generated method stub

}



class MIDIFileFilter extends FileFilter {
```

```
            public String getDescription() {
                return "MIDI files (*.midi)";
            }

            public boolean accept(File f) {
                return f.isDirectory() || f.getName().endsWith(".midi");
            }
        }

}
```

==========

==========

```
package diamouses.ui.scoreEditor;

import diamouses.ui.scoreEditor.staves.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.net.URL;
import java.util.Vector;
import javax.swing.*;
import javax.swing.border.*;

import jm.music.data.Note;
import jm.music.data.Part;
import jm.music.data.Phrase;
import jm.util.Play;

public class ScoreEditor extends JPanel implements
ActionListener {

      /** Private Variables **/
      private JComboBox stave_selection_combobox;
      protected DStave stave;
      private JPanel scorePanel;
      private JScrollPane scroll;
      private String selectedInstrument = "Guitar (Clean)       027";
      private JButton remove_last_note_button,play_button ;
      /**
       * Constructor Creates new form ScoreEditor
       **/
      public ScoreEditor() {

            Phrase ph = new Phrase();

            /*Vector<Note> notes = new Vector<Note>();
            for (int i = 0; i < 40; i++) {
                  double min = 41;
                  double max = 78;
                  double pitch = (max - min) * Math.random() + min;
                  Note note1 = new Note((int) pitch, 2 * Math.random());
                  notes.add(note1);
            }
```

```
        ph.addNoteList(notes, true);

        DGrandStave dg_stave = new DGrandStave(ph);
        */
        DGrandStave dg_stave = new DGrandStave();
        stave = dg_stave;
        stave.setTitle(selectedInstrument);
        stave.setDisplayTitle(true);

        scorePanel = new JPanel();
        scorePanel.setBackground(Color.getHSBColor((float) 0.14,
           (float) 0.09,(float) 1.0)); // .17, .1, 1

        scorePanel.add(stave, BorderLayout.CENTER);
        scroll = new JScrollPane(stave);
        scroll.setViewportView(scorePanel);

        setLayout(new BorderLayout());
        add(scroll, BorderLayout.CENTER);

        stave_selection_combobox = new JComboBox();
        stave_selection_combobox.setModel(new
        DefaultComboBoxModel(new String[] {"Grand Stave", "Treble
        Stave", "Bass Stave" }));
        stave_selection_combobox.setActionCommand("Select_Stave");
        stave_selection_combobox.addActionListener(this);
        stave_selection_combobox.setSelectedItem(1);
        selectStave();
}
/**
* Method creates a "smart" JToggleButton
* @param imageName The location of the image of this toggle-button
* @param selected_imageName The location of the image of this
toggle-button
* when the toggle button is selected
* @param actionCommand The action-command to set for this
components
* @param toolTipText the tooltip of the component
* @param altText Alternative text in case the image is missing
* @return A JToggleButton component
*/
protected JToggleButton makeToggleButton(String imageName, String
selected_imageName,String actionCommand, String toolTipText, String
altText) {
        // Look for the image.
        String imgLocation = "images/menu_images/" + imageName +
        ".gif";
        String imgLocation2 = "images/menu_images/" +
        selected_imageName + ".gif";

        URL imageURL = Main.class.getResource(imgLocation);
        URL imageURL2 = Main.class.getResource(imgLocation2);

        // Create and initialize the button.
        JToggleButton button = new JToggleButton();
        button.setActionCommand(actionCommand);
        button.setToolTipText(toolTipText);
```

```
        button.addActionListener(this);

        if ( (imageURL != null) & (imageURL2 != null) ){ // image
        found
              button.setIcon(new ImageIcon(imageURL, altText));
              button.setSelectedIcon(new ImageIcon(imageURL2,
              altText));
        } else { // no image found
              button.setText(altText);
              System.err.println("Resource not found: " + imgLocation
              + "" + (imageURL==null) + " " + (imageURL2==null));
        }

        return button;
}

/**
* Method creates a "smart" JButton
* @param imageName The location of the image of this button
* @param actionCommand The action-command to set for this
components
* @param toolTipText the tooltip of the component
* @param altText Alternative text in case the image is missing
* @return A JButton component
*/
protected JButton makeButton(String imageName, String
actionCommand, String toolTipText, String altText) {
        // Look for the image.
        String imgLocation = "images/menu_images/" + imageName +
        ".gif";

        URL imageURL = Main.class.getResource(imgLocation);

        // Create and initialize the button.
        JButton button = new JButton();
        button.setActionCommand(actionCommand);
        button.setToolTipText(toolTipText);
        button.setText(toolTipText);
        button.addActionListener(this);

        if  (imageURL != null) { // image found
              button.setIcon(new ImageIcon(imageURL, altText));
        } else { // no image found
              button.setText(altText);
              System.err.println("Resource not found 2: " +
              imgLocation);
        }
        return button;
}

/**
* Method creates and returns the top part of the Score-Editor with
the buttons
* @return the JPanel created with control buttons
*/

public JPanel getJToolBar() {
```

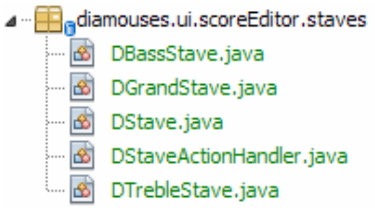
```
// 1. Create the control panel
JButton new_stave, metronome_button;
new_stave = makeButton("clearscore", "Clear_Stave", "Clear
Stave","altText");
play_button = makeButton("play", "Play", "Play", "altText");
remove_last_note_button = makeButton("removelastnote",
"Remove_Last_Note","Remove Note", "altText");
remove_last_note_button.setEnabled(false);
play_button.setEnabled(false);

JPanel control_panel = new JPanel();
control_panel.setBorder(BorderFactory.createTitledBorder(null
, "Control Buttons", TitledBorder.DEFAULT_JUSTIFICATION,
TitledBorder.DEFAULT_POSITION, new Font("Tahoma", 0, 10)));
control_panel.add(new_stave);
control_panel.add(play_button);
control_panel.add(remove_last_note_button);

// 2. Create the NOTE selection panel
JPanel note_panel = new JPanel();
note_panel.setBorder(BorderFactory.createTitledBorder(null,
"Select Note", TitledBorder.DEFAULT_JUSTIFICATION,
TitledBorder.DEFAULT_POSITION, new Font("Tahoma", 0, 10)));

JToggleButton note16th_button, note_8th_button,
note_4th_button,note_Half_button, note_Full_button;

note16th_button =
makeToggleButton("semiquaverUp","semiquaverUp_selected",
"Note_16th","Semi quaver Up", "altText");

note_8th_button =
makeToggleButton("quaverUp","quaverUp_selected", "Note_8th",
"Quaver Up", "altText");

note_4th_button = makeToggleButton("crotchetUp",
"crotchetUp_selected", "Note_4th","Crotchet Up", "altText");

note_Half_button = makeToggleButton("minimUp",
"minimUp_selected", "Note_Half","Minim Up", "altText");
note_Full_button = makeToggleButton("semibreve",
"semibreve_selected","Whole","Semi Breve", "altText");

note_panel.add(note16th_button);
note_panel.add(note_8th_button);
note_panel.add(note_4th_button);
note_panel.add(note_Half_button);
note_panel.add(note_Full_button);

// 3. Create the REST selection panel
JPanel rest_panel = new JPanel();
rest_panel.setBorder(BorderFactory.createTitledBorder(null,
"Add Rest", TitledBorder.DEFAULT_JUSTIFICATION,
TitledBorder.DEFAULT_POSITION, new Font("Tahoma", 0, 10)));
```

```
JToggleButton semi_quaver_rest_button, quaver_rest_button,
crotchet_rest_button,minim_rest_button,
semibreve_rest_button;

semi_quaver_rest_button =
makeToggleButton("semiquaverRest","semiquaverRest",
"Rest_16th","Semi quarter Rest", "altText");

quaver_rest_button = makeToggleButton("quaverRest",
"quaverRest","Rest_8th","Quarter Rest", "altText");

crotchet_rest_button = makeToggleButton("crotchetRest",
"crotchetRest","Rest_4th","Crotchet Rest", "altText");

minim_rest_button = makeToggleButton("minimRest",
"minimRest","Rest_Half", "Minim Rest","altText");

semibreve_rest_button = makeToggleButton("semibreveRest",
"semibreveRest", "Rest_Whole","Semi breve Rest", "altText");

rest_panel.add(semi_quaver_rest_button);
rest_panel.add(quaver_rest_button);
rest_panel.add(crotchet_rest_button);
rest_panel.add(minim_rest_button);
rest_panel.add(semibreve_rest_button);

// 4. Only one Note or one Rest can be selected at a time,
// So add all toggle buttons in a button group
ButtonGroup group = new ButtonGroup();
group.add(note16th_button);
group.add(note_8th_button);
group.add(note_4th_button);
group.add(note_Half_button);
group.add(note_Full_button);
group.add(semi_quaver_rest_button);
group.add(quaver_rest_button);
group.add(crotchet_rest_button);
group.add(minim_rest_button);
group.add(semibreve_rest_button);

// 5. Add Instrument selection panel
JPanel instrument_selection = new JPanel();
instrument_selection.setBorder(BorderFactory.createTitledBord
er(null, "Options", TitledBorder.DEFAULT_JUSTIFICATION,
TitledBorder.DEFAULT_POSITION, new Font("Tahoma", 0, 10)));

JButton select_instrumnet_button =
makeButton("selectinstrument", "Show_Instrument_Dialog",
                        "Select Instrument", "altText");

instrument_selection.add(stave_selection_combobox);
instrument_selection.add(select_instrumnet_button);

JPanel main_panel = new JPanel();
main_panel.setLayout(new
BoxLayout(main_panel,BoxLayout.X_AXIS));
main_panel.add(control_panel);
```

```
        main_panel.add(note_panel);
        main_panel.add(rest_panel);
        main_panel.add(instrument_selection);

        return main_panel;
}

/**
 * Method to listen and react to actions
 */
public void actionPerformed(ActionEvent e) {
        // Get the action-command associated with the event
        String action_command = e.getActionCommand();
        if (action_command.equals("Note_16th"))
                stave.setNoteDuration(0.25);
        else if (action_command.equals("Note_8th"))
                stave.setNoteDuration(0.5);
        else if (action_command.equals("Note_4th"))
                stave.setNoteDuration(1.0);
        else if (action_command.equals("Note_Half"))
                stave.setNoteDuration(2.0);
        else if (action_command.equals("Note_Whole"))
                stave.setNoteDuration(4.0);
        else if (action_command.equals("Rest_16th"))
                setRest(0.25);
        else if (action_command.equals("Rest_8th"))
                setRest(0.5);
        else if (action_command.equals("Rest_4th"))
                setRest(1.0);
        else if (action_command.equals("Rest_Half"))
                setRest(2.0);
        else if (action_command.equals("Rest_Whole"))
                setRest(4.0);
        else if (action_command.equals("Select_Instrument"))
                selectStave();
        else if (action_command.equals("Clear_Stave")) {
                remove_last_note_button.setEnabled(false);
                play_button .setEnabled(false);
                stave.setPhrase(new Phrase());
        }
        else if (action_command.equals("Remove_Last_Note"))
                removeLastNote();
        else if (action_command.equals("Show_Instrument_Dialog"))
                showInstrumentDialog();
        else if (action_command.equals("Play"))
                playTune();
}

public Part getPart() {
        Phrase phr = stave.getPhrase();
        Part part = new Part(phr);
        String s = selectedInstrument;
        s = s.substring(21, 24);
        s.trim();

        part.setInstrument(Integer.valueOf(s));
```

```
		return part;
	}


	/**
	 * Method that plays the tune in the stave
	 */
	protected void playTune() {

		Play.midi(getPart(), false);
	}

	/**
	 * Method that shows an Instrument Selection Dialog
	 */
	private void showInstrumentDialog() {
		Object result = (Object) JOptionPane.showInputDialog(this,
		"","Select instrument", JOptionPane.PLAIN_MESSAGE, null,
		initializeInstrumentList(), selectedInstrument);
		String s = null;
		if (result instanceof String)
			s = (String) result;
		if (s != null) {
			selectedInstrument = s;
			stave.setTitle(s);
			stave.repaint();
		}
	}
	/**
	 * Method that removes last note in the stave
	 */
	private void removeLastNote() {
		Phrase phrase = stave.getPhrase();
		if (phrase.getSize()>0) {
			phrase.removeLastNote();
			stave.setPhrase(phrase);
			if (phrase.getSize()==0) {
				remove_last_note_button.setEnabled(false);
				play_button.setEnabled(false);
			}
		}
	}

	/**
	 * Method that selects the stave
	 */
	private void selectStave() {
		Phrase phr = stave.getPhrase();
		String item = (String)
		stave_selection_combobox.getSelectedItem();
		scorePanel.remove(stave);
		if (item.equalsIgnoreCase("Grand Stave")) {
			stave = new DGrandStave(phr);
			scorePanel.add(stave, BorderLayout.CENTER);
		} else if (item.equalsIgnoreCase("Treble Stave")) {
			stave = new DTrebleStave(phr);
			scorePanel.add(stave, BorderLayout.CENTER);
```

```
            } else if (item.equalsIgnoreCase("Bass Stave")) {
                  stave = new DTrebleStave(phr);
                  scorePanel.add(stave, BorderLayout.CENTER);
            }

            stave.setTitle(selectedInstrument);
            stave.setDisplayTitle(true);
            scorePanel.repaint();
      }

      /**
       * This method adds a new Rest if no Note is selected,
       * or changes the selected note to a rest.
       *
       * @param rest_value Rest value is the value of the rest, in the
       range:
       * 4.0 (whole-rest) 2.0 (half-rest) 1.0 (quarter-rest)
       * 0.5 (eight-rest) 0.25 (sixteenth-rest)
       */
      private void setRest(double rest_value) {
            Phrase phr = stave.getPhrase();
            Note n;
            if (stave.getSelectedNote() == -1) {
                  n = new Note(-2147483648, rest_value);
                  phr.add(n);
            } else {
                  n = phr.getNote(stave.getSelectedNote());
                  n.setPitch(-2147483648);
                  n.setRhythmValue(rest_value);
            }
            remove_last_note_button.setEnabled(true);
            stave.repaint();
      }
      /**
       * Method returns an array of valid Guitar instruments
       */
      private String [] initializeInstrumentList() {
            String [] instrumentList = {
                        "Guitar (Clean)       027",
                        "Guitar (Distorted)   030",
                        "Guitar Harmonics     031",
                        "Guitar (Jazz)        026",
                        "Guitar (Muted)       028",
                        "Guitar (Nylon)       024",
                        "Guitar (Overdrive)   029",
                        "Guitar (Steel)       025" };
            return instrumentList;
      }

      public void setEnabledRemove(boolean b) {
            remove_last_note_button.setEnabled(b);
            play_button.setEnabled(b);
      }
}
```




```
package diamouses.ui.scoreEditor.staves;

import java.awt.*;
import java.awt.image.BufferedImage;

import diamouses.ui.scoreEditor.GuitarNeck;

import jm.JMC;
import jm.music.data.*;

  public class DBassStave extends DStave implements JMC {

      public DBassStave() {
          super();
          staveDelta = staveSpaceHeight*11/2;
      }

      public DBassStave(Phrase phrase) {
          super(phrase);
          staveDelta = staveSpaceHeight*11/2;
      }
       public void paintComponent(Graphics graphics) {
          // set up for double buffering
        if (image == null) {
            image = this.createImage(this.getSize().width,
            this.getSize().height);
            g = image.getGraphics();
          }
          g.setFont(font);
          // keep track of the rhythmic values for bar lines
          double beatCounter = 0.0;
          // reset the chromatic vector
          previouslyChromatic.removeAllElements();
          // reste note position locations
          notePositions.removeAllElements();
          int keyAccidentals = 0;
          // add a title if set to be visible
          if (getDisplayTitle()) {
            g.drawString(title, rightMargin, bPos - 10);
        // insert key signature if required
          }
          int keyOffset = 0;
          // is the key signature using sharps or flats?
          if (keySignature > 0 && keySignature < 8) { // sharp
              for (int ks = 0; ks < keySignature; ks++) {
                  // claulate position
```

```
                int keyAccidentalPosition = notePosOffset[sharps[ks] %
                12] + bPos - 4 + ((5 - sharps[ks] / 12) * 24) + ((6 -
                sharps[ks] / 12) * 4);
                // draw sharp
                g.drawImage(sharp, rightMargin + clefWidth + keyOffset,
                keyAccidentalPosition + staveSpaceHeight , this);
                 // indent position
                keyOffset += 10;
                //add note to accidental vector
                int theModValue = sharps[ks] % 12;
                for (int pc = 0; pc < 128; pc++) {
                    if ((pc % 12) == theModValue) {
                        previouslyChromatic.addElement(new Integer(pc));
                        keyAccidentals++;
                    }
                }
                keySigWidth = keyOffset;
            }
        } else {
            if (keySignature < 0 && keySignature > -8) { // flat
                for (int ks = 0; ks < Math.abs(keySignature); ks++) {
                    // claulate position
                    int keyAccidentalPosition = notePosOffset[flats[ks] %
                    12] + bPos - 4 + ((5 - flats[ks] / 12) * 24) + ((6 -
                    flats[ks] / 12) * 4);
                    // draw flat
                    g.drawImage(flat, rightMargin + clefWidth +
                    keyOffset, keyAccidentalPosition + staveSpaceHeight,
                    this);
                     // indent position
                     keyOffset += 10;

                //add note to accidental vector
                int theModValue = sharps[ks] % 12;
                for (int pc = 0; pc < 128; pc++) {
                    if ((pc % 12) == theModValue) {
                        previouslyChromatic.addElement(new Integer(pc));
                        keyAccidentals++;
                    }
                }

keySigWidth = keyOffset;
            }
        } else {
            if (keySignature < 0 && keySignature > -8) { // flat
                for (int ks = 0; ks < Math.abs(keySignature); ks++) {
                    // claulate position
                    int keyAccidentalPosition = notePosOffset[flats[ks] %
                    12] + bPos - 4 + ((5 - flats[ks] / 12) * 24) + ((6 -
                    flats[ks] / 12) * 4);
                    // draw flat
                    g.drawImage(flat, rightMargin + clefWidth +
                    keyOffset, keyAccidentalPosition + staveSpaceHeight,
                    this);
                     // indent position
                     keyOffset += 10;
                     //add note to accidental vector
```

```
                int theModValue = flats[ks] % 12;
                for (int pc = 0; pc < 128; pc++) {
                    if ((pc % 12) == theModValue) {

                        previouslyChromatic.addElement(new
                        Integer(pc));
                        keyAccidentals++;
                    }
                }
            }
        }
    }
    keySigWidth = keyOffset + 3;

    // insert time signature if required
    if (metre != 0.0) {
        Image[] numbers = {one, two, three, four, five, six, seven,
        eight, nine};

        // top number
        g.drawImage(numbers[(int) metre - 1], rightMargin + clefWidth
        + keySigWidth, bPos + 13, this);
        //bottom number
        g.drawImage(four, rightMargin + clefWidth + keySigWidth, bPos
        + 29, this);
        timeSigWidth = 30;
    } else  timeSigWidth = 5;
    // set indent position for first note
    totalBeatWidth = rightMargin + clefWidth + keySigWidth +
    timeSigWidth;

     // draw notes and rests
    for (int i = 0; i < phrase.size(); i++) {
        int notePitchNum = (int) phrase.getNote(i).getPitch();
        // choose graphic
        chooseImage(notePitchNum, phrase.getNote(i).getRhythmValue(),
        50, 0, 50);
        // reset pitch for rests

        // position?
        int pitchTempPos;
        if (notePitchNum == REST ||
        phrase.getNote(i).getRhythmValue() == 0.0) { // rest or
        delete
        pitchTempPos = notePosOffset[71 % 12] + bPos - 4 + ((5 - 71 /
        12) * 24) + ((6 - 71 / 12) * 4);
        } else {
            pitchTempPos = notePosOffset[notePitchNum % 12] + bPos -
            4 + ((5 - notePitchNum / 12) * 24) + ((6 - notePitchNum
            / 12) * 4 - staveSpaceHeight * 6);
        }

        // accidental?
        if (((notePitchNum % 12) == 1 || (notePitchNum % 12) == 3 ||
        (notePitchNum % 12) == 6 || (notePitchNum % 12) == 8 ||
        (notePitchNum % 12) == 10) && notePitchNum != REST &&
        phrase.getNote(i).getRhythmValue() != 0.0) {
```

```
            if (keySignature > -1) {
                g.drawImage(sharp, totalBeatWidth - 9, pitchTempPos,
                this);
                previouslyChromatic.addElement(new
                Integer(notePitchNum - 1)); // enter the note made
                sharp i.e, F for an F#
            } else { // flat
                pitchTempPos -= 4; // to show the note a semitone
                higher for flats
                g.drawImage(flat, totalBeatWidth - 9, pitchTempPos,
                this);
                previouslyChromatic.addElement(new
                Integer(notePitchNum + 1));
                notePitchNum++; // assume it is a semitone higher for
                legerlines etc...
            }
        } else { // check for a natural
            // check vector
            int size = previouslyChromatic.size();
            for (int j = 0; j < size; j++) {
                Integer temp = (Integer)
                previouslyChromatic.elementAt(j);
                if (temp.intValue() == notePitchNum && notePitchNum
                != REST && phrase.getNote(i).getRhythmValue() != 0.0)
                {
                     // add natural
                     g.drawImage(natural, totalBeatWidth - 7,
                     pitchTempPos, this);

                     // remove element if not in key signature
                     if (j > keyAccidentals - 1) {
                         previouslyChromatic.removeElementAt(j);
                     }
                     j = size;
                 }
            }
        }

        // draw note/rest
        if (currImage != null) {
              if (i==getSelectedNote()) {
                    Note selected_note =
                    phrase.getNote(i);//.getPitch();

GuitarNeck.getInstance().setNote(selected_note.getPitch());
             //-System.out.println(selected_note.toString());
                    BufferedImage bi =
                    DStave.toBufferedImage(currImage);
                    int w = bi.getWidth();
                    int h = bi.getHeight();
                    int pixel;
                    BufferedImage biOut = new BufferedImage(w, h,
                    bi.TYPE_INT_ARGB_PRE);

                    for (int x = 0; x < w; x++) {
                          for (int y = 0; y < h; y++) {
                                pixel = bi.getRGB(x, y);
```

```
                                if (pixel == Color.BLACK).getRGB()) {
                                      pixel = (Color.RED).getRGB();
                                }
                                biOut.setRGB(x, y, pixel);
                          }
                    }

                    currImage = DStave.toImage(biOut);
              }
        }

g.drawImage(currImage, totalBeatWidth, pitchTempPos, this);
        // store position in a vector
        notePositions.addElement(new Integer(totalBeatWidth));
         notePositions.addElement(new Integer(pitchTempPos +
         staveDelta)); // stave delta required for bass clef offset
         from treble
         //System.out.println("Position "+i+" "+totalBeatWidth + "
         "+ pitchTempPos);

        if (dottedNote) {
            boolean dotFlag = true;
            for (int l = 0; l < lineNotes.length; l++) {
                if (lineNotes[l] + 12 == notePitchNum || lineNotes[l]
                + 36 == notePitchNum || lineNotes[l] + 60 ==
                notePitchNum || lineNotes[l] + 84 == notePitchNum ||
                lineNotes[l] + 108 == notePitchNum || notePitchNum ==
                REST) {
                     g.drawImage(dot, totalBeatWidth + 1, pitchTempPos
                     - 4, this);
                     dotFlag = false;
                     l = lineNotes.length;
                 }
            }
            if (dotFlag) {
                g.drawImage(dot, totalBeatWidth + 1, pitchTempPos,
                this);
            }
        }

        // leger lines down
        if (notePitchNum <= 40 && notePitchNum > -1 &&
        phrase.getNote(i).getRhythmValue() != 0.0) {
        g.drawLine(totalBeatWidth - 3, bPos + 52, totalBeatWidth +
        12, bPos + 52);
        }
        if (notePitchNum <= 37 && notePitchNum > -1 &&
        phrase.getNote(i).getRhythmValue() != 0.0) {
        g.drawLine(totalBeatWidth - 3, bPos + 60, totalBeatWidth +
        12, bPos + 60);
        }
        if (notePitchNum <= 34 && notePitchNum > -1 &&
        phrase.getNote(i).getRhythmValue() != 0.0) {
        g.drawLine(totalBeatWidth - 3, bPos + 68, totalBeatWidth +
        12, bPos + 68);
        }
```

```
if (notePitchNum <= 30 && notePitchNum > -1 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos + 76, totalBeatWidth +
12, bPos + 76);
}
if (notePitchNum <= 26 && notePitchNum > -1 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos + 84, totalBeatWidth +
12, bPos + 84);
}

// leger lines up
if (notePitchNum >= 60 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos + 4, totalBeatWidth + 12,
bPos + 4);
}
if (notePitchNum >= 64 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos - 4, totalBeatWidth + 12,
bPos - 4);
}
if (notePitchNum >= 67 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos - 12, totalBeatWidth +
12, bPos - 12);
}


if (notePitchNum >= 71 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos - 20, totalBeatWidth +
12, bPos - 20);
}
if (notePitchNum >= 74 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos - 28, totalBeatWidth +
12, bPos - 28);

// increment everything
totalBeatWidth += currBeatWidth;
dottedNote = false;
// quantised to semiquvers!
// (int)((phrase.getNote(i).getRhythmValue()/0.25) * 0.25);
beatCounter += (int) (phrase.getNote(i).getRhythmValue() /
0.25) * 0.25;


// add bar line if required
if (metre != 0.0) {
    if ((beatCounter % metre) == 0.0) {
        g.drawLine(totalBeatWidth, bPos + 12, totalBeatWidth,
        bPos + 44);
         // add bar numbers?
         if (barNumbers) {
```

```
                        g.drawString("" + (int) (beatCounter / metre + 1
                        + phrase.getStartTime()), totalBeatWidth - 4,
                        bPos);
                    }
                    totalBeatWidth += 12;
                }
            }
    }
        // draw stave
        for (int i = 0; i < 5; i++) {
            g.drawLine(rightMargin, (bPos + imageHeightOffset - (2 *
            staveSpaceHeight)) + (i * staveSpaceHeight), totalBeatWidth,
            (bPos + imageHeightOffset - (2 * staveSpaceHeight));
        }
        // draw next note stave area
        // draw stave
        g.setColor(Color.lightGray);

        for (int i = 0; i < 5; i++) {
            g.drawLine(totalBeatWidth,
            (bPos + imageHeightOffset - (2 * staveSpaceHeight)) + (i *
            staveSpaceHeight),totalBeatWidth + 50,(bPos +
            imageHeightOffset - (2 * staveSpaceHeight)) + (i *
            staveSpaceHeight));
    }
        g.setColor(Color.black);
        // add Clef
        g.drawImage(bassClef, rightMargin + 7, bPos, this);

        /* Draw completed buffer to g */
        graphics.drawImage(image, 0, 0, null);
        // clear image
        // clear
        g.setColor(this.getBackground());
        g.fillRect(0, 0, getSize().width, getSize().height);
        g.setColor(this.getForeground());
    //repaint();
    //g.dispose();
    }
}
```

```
================================================================================
================================================================================
```

```
package diamouses.ui.scoreEditor.staves;

import java.awt.*;
import java.awt.image.BufferedImage;

import diamouses.ui.scoreEditor.GuitarNeck;

import jm.JMC;
import jm.music.data.*;

public class DGrandStave extends DStave implements JMC {
```

```
public DGrandStave() {
    super();
    bPos = 110;
    panelHeight = 310;
    this.setSize((int) (beatWidth * spacingValue), panelHeight);
}

public DGrandStave(Phrase phrase) {
    super(phrase);
    bPos = 110;
    panelHeight = 310;
    this.setSize((int) (beatWidth * spacingValue), panelHeight);
}

public void paintComponent(Graphics graphics) {
    // set up for double buffering
    if (image == null) {
        image = this.createImage(this.getSize().width,
        this.getSize().height);
        g = image.getGraphics();
    }
    g.setFont(font);
    // keep track of the rhythmic values for bar lines
    double beatCounter = 0.0;
    // reset the chromatic vector
    previouslyChromatic.removeAllElements();
    // reste note position locations
    notePositions.removeAllElements();
    int keyAccidentals = 0;
    // add a title if set to be visible
    if (getDisplayTitle()) {
        g.drawString(title, rightMargin, 60); //bPos - 10);
    // insert key signature if required
    }
    int keyOffset = 0;
    // is the key signature using sharps or flats?
    if (keySignature > 0 && keySignature < 8) { // sharp
        for (int ks = 0; ks < keySignature; ks++) {
            // claulate position
            int keyAccidentalPosition = notePosOffset[sharps[ks] %
            12] + bPos - 4 + ((5 - sharps[ks] / 12) * 24) + ((6 -
            sharps[ks] / 12) * 4);
            // draw sharp on treble
            g.drawImage(sharp, rightMargin + clefWidth + keyOffset,
            keyAccidentalPosition, this);
            // draw sharp on bass
            g.drawImage(sharp, rightMargin + clefWidth + keyOffset,
            keyAccidentalPosition + staveSpaceHeight * 7, this);
            // indent position
            keyOffset += 10;
            //add note to accidental vector
            int theModValue = sharps[ks] % 12;
            for (int pc = 0; pc < 128; pc++) {
                if ((pc % 12) == theModValue) {
                    previouslyChromatic.addElement(new Integer(pc));
                    keyAccidentals++;
```

```
                }
            }
            keySigWidth = keyOffset;
        }
    } else {
        if (keySignature < 0 && keySignature > -8) { // flat
            for (int ks = 0; ks < Math.abs(keySignature); ks++) {
                // claulate position
                int keyAccidentalPosition = notePosOffset[flats[ks] %
                12] + bPos - 4 + ((5 - flats[ks] / 12) * 24) + ((6 -
                flats[ks] / 12) * 4);
                // draw flat
                g.drawImage(flat, rightMargin + clefWidth +
                keyOffset, keyAccidentalPosition, this);
                // draw flat on bass stave
                g.drawImage(flat, rightMargin + clefWidth +
                keyOffset, keyAccidentalPosition + staveSpaceHeight *
                7, this);
                // indent position
                keyOffset += 10;
                //add note to accidental vector
                int theModValue = flats[ks] % 12;
                for (int pc = 0; pc < 128; pc++) {
                    if ((pc % 12) == theModValue) {

                        previouslyChromatic.addElement(new
                        Integer(pc));
                        keyAccidentals++;
                    }
                }
            }
        }
    }
}
keySigWidth = keyOffset + 3;

// insert time signature if required
if (metre != 0.0) {
    Image[] numbers = {one, two, three, four, five, six, seven,
    eight, nine};

    // top number
    g.drawImage(numbers[(int) metre - 1], rightMargin + clefWidth
    + keySigWidth, bPos + 13, this);
    g.drawImage(numbers[(int) metre - 1], rightMargin + clefWidth
    + keySigWidth, bPos + 13 + staveSpaceHeight * 6, this);
    //bottom number
    g.drawImage(four, rightMargin + clefWidth + keySigWidth, bPos
    + 29, this);
    g.drawImage(four, rightMargin + clefWidth + keySigWidth, bPos
    + 29 + staveSpaceHeight * 6, this);
    timeSigWidth = 30;
} else {
    timeSigWidth = 5;
// set indent position for first note
}
totalBeatWidth = rightMargin + clefWidth + keySigWidth +
timeSigWidth;
```

```
// draw notes and rests
for (int i = 0; i < phrase.size(); i++) {
    int notePitchNum = (int) phrase.getNote(i).getPitch();
    // choose graphic
    chooseImage(notePitchNum, phrase.getNote(i).getRhythmValue(),
    71, 60, 50);
    // reset pitch for rests

    // position?
    int pitchTempPos;
    if (notePitchNum == REST ||
    phrase.getNote(i).getRhythmValue() == 0.0) { // rest or
    delete
    pitchTempPos = notePosOffset[71 % 12] + bPos - 4 + ((5 - 71 /
    12) * 24) + ((6 - 71 / 12) * 4);
    } else {
        pitchTempPos = notePosOffset[notePitchNum % 12] + bPos -
        4 + ((5 - notePitchNum / 12) * 24) + ((6 - notePitchNum
        / 12) * 4);
    }

    // accidental?
    if (((notePitchNum % 12) == 1 || (notePitchNum % 12) == 3 ||
    (notePitchNum % 12) == 6 || (notePitchNum % 12) == 8 ||
    (notePitchNum % 12) == 10) && notePitchNum != REST &&
    phrase.getNote(i).getRhythmValue() != 0.0) {
        if (keySignature > -1) {
            g.drawImage(sharp, totalBeatWidth - 9, pitchTempPos,
            this);
            previouslyChromatic.addElement(new
            Integer(notePitchNum - 1)); // enter the note made
            sharp i.e, F for an F#
        } else { // flat
            pitchTempPos -= 4; // to show the note a semitone
            higher for flats
            g.drawImage(flat, totalBeatWidth - 9, pitchTempPos,
            this);
            previouslyChromatic.addElement(new
            Integer(notePitchNum + 1));
            notePitchNum++; // assume it is a semitone higher for
            legerlines etc...
        }
    } else { // check for a natural
        // check vector
        int size = previouslyChromatic.size();
        for (int j = 0; j < size; j++) {
            Integer temp = (Integer)
            previouslyChromatic.elementAt(j);
            if (temp.intValue() == notePitchNum && notePitchNum
            != REST && phrase.getNote(i).getRhythmValue() != 0.0)
            {
                // add natural
                g.drawImage(natural, totalBeatWidth - 7,
                pitchTempPos, this);
                // remove element if not in key signature
                if (j > keyAccidentals - 1) {
```

```
                        previouslyChromatic.removeElementAt(j);
                    }
                    j = size;
                }
            }
        }

        /* draw the selection line
        if ((i >= getSelectedNote()) && (i <= getSelectedNote() +
        getSelectedArea())) {
        g.setColor(new Color(255, 0, 0));
        g.drawLine(totalBeatWidth, 10, totalBeatWidth +
        currImage.getWidth(this), 10);
        g.drawLine(totalBeatWidth, 10, totalBeatWidth, 12);
        g.drawLine(totalBeatWidth + currImage.getWidth(this), 10,
        totalBeatWidth + currImage.getWidth(this), 12);
        g.setColor(new Color(0, 0, 0));
        }
        */

        // draw note/rest
        if (currImage != null) {
              if (i==getSelectedNote()) {
                    Note selected_note =
                    phrase.getNote(i);//.getPitch();

GuitarNeck.getInstance().setNote(selected_note.getPitch());
                    //-System.out.println(selected_note.toString());
                    BufferedImage bi =
                    DStave.toBufferedImage(currImage);
                    int w = bi.getWidth();
                    int h = bi.getHeight();
                    int pixel;
                    BufferedImage biOut = new BufferedImage(w, h,
                    bi.TYPE_INT_ARGB_PRE);

                    for (int x = 0; x < w; x++) {
                          for (int y = 0; y < h; y++) {
                                pixel = bi.getRGB(x, y);
                                if (pixel == (Color.BLACK).getRGB())
{
                                      pixel = (Color.RED).getRGB();
                                }
                                biOut.setRGB(x, y, pixel);
                          }
                    }

                    currImage = DStave.toImage(biOut);
              }
        }
        g.drawImage(currImage, totalBeatWidth, pitchTempPos, this);
        // store position in a vector
        notePositions.addElement(new Integer(totalBeatWidth));
        notePositions.addElement(new Integer(pitchTempPos));

        if (dottedNote) {
            boolean dotFlag = true;
```

```
        for (int l = 0; l < lineNotes.length; l++) {
            if (lineNotes[l] + 12 == notePitchNum || lineNotes[l]
            + 36 == notePitchNum || lineNotes[l] + 60 ==
            notePitchNum || lineNotes[l] + 84 == notePitchNum ||
            lineNotes[l] + 108 == notePitchNum || notePitchNum ==
            REST) {
                 g.drawImage(dot, totalBeatWidth + 1, pitchTempPos
                 - 4, this);
                 dotFlag = false;
                 l = lineNotes.length;
             }
        }
        if (dotFlag) {
            g.drawImage(dot, totalBeatWidth + 1, pitchTempPos,
            this);
        }
    }
    // leger lines middle C
    if (notePitchNum == 60 || notePitchNum == 61 &&
    phrase.getNote(i).getRhythmValue() != 0.0) {
    g.drawLine(totalBeatWidth - 3, bPos + 52, totalBeatWidth +
    12, bPos + 52);
    }

    // leger lines down
    if (notePitchNum <= 40 && notePitchNum > -1 &&
    phrase.getNote(i).getRhythmValue() != 0.0) {
    g.drawLine(totalBeatWidth - 3, bPos + 100, totalBeatWidth +
    12, bPos + 100);
    }
    if (notePitchNum <= 37 && notePitchNum > -1 &&
    phrase.getNote(i).getRhythmValue() != 0.0) {
    g.drawLine(totalBeatWidth - 3, bPos + 108, totalBeatWidth +
    12, bPos + 108);
    }
    // leger lines down low
    if (notePitchNum <= 16 && notePitchNum > -1 &&
    phrase.getNote(i).getRhythmValue() != 0.0) {
    g.drawLine(totalBeatWidth - 3, bPos + 156, totalBeatWidth +
    12, bPos + 156);
    }
    if (notePitchNum <= 13 && notePitchNum > -1 &&
    phrase.getNote(i).getRhythmValue() != 0.0) {
    g.drawLine(totalBeatWidth - 3, bPos + 164, totalBeatWidth +
    12, bPos + 164);
    }
    if (notePitchNum <= 10 && notePitchNum > -1 &&
    phrase.getNote(i).getRhythmValue() != 0.0) {
    g.drawLine(totalBeatWidth - 3, bPos + 172, totalBeatWidth +
    12, bPos + 172);
    }
    if (notePitchNum <= 6 && notePitchNum > -1 &&
    phrase.getNote(i).getRhythmValue() != 0.0) {
    g.drawLine(totalBeatWidth - 3, bPos + 180, totalBeatWidth +
    12, bPos + 180);
    }
```

```
if (notePitchNum <= 3 && notePitchNum > -1 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos + 188, totalBeatWidth +
12, bPos + 188);
}

// leger lines up
if (notePitchNum >= 81 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos + 4, totalBeatWidth + 12,
bPos + 4);
}
if (notePitchNum >= 84 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos - 4, totalBeatWidth + 12,
bPos - 4);
}
// leger lines up high
if (notePitchNum >= 105 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos - 52, totalBeatWidth +
12, bPos - 52);
}
if (notePitchNum >= 108 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos - 60, totalBeatWidth +
12, bPos - 60);
}
if (notePitchNum >= 112 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos - 68, totalBeatWidth +
12, bPos - 68);
}
if (notePitchNum >= 115 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos - 76, totalBeatWidth +
12, bPos - 76);
}
if (notePitchNum >= 119 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos - 84, totalBeatWidth +
12, bPos - 84);
}
if (notePitchNum >= 122 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos - 92, totalBeatWidth +
12, bPos - 92);
}
if (notePitchNum >= 125 && notePitchNum < 128 &&
phrase.getNote(i).getRhythmValue() != 0.0) {
g.drawLine(totalBeatWidth - 3, bPos - 100, totalBeatWidth +
12, bPos - 100);
}

// increment everything
totalBeatWidth += currBeatWidth;
dottedNote = false;
```

```
        // quantised to semiquvers!
        beatCounter += (int) (phrase.getNote(i).getRhythmValue() /
        0.25) * 0.25;

        // add bar line if required
        if (metre != 0.0) {
            if ((beatCounter % metre) == 0.0) {
                g.drawLine(totalBeatWidth, bPos + 12 -
                staveSpaceHeight * 7, totalBeatWidth, bPos + 44 +
                staveSpaceHeight * 13);
                // add bar numbers?
                if (barNumbers) {
                    g.drawString("" + (int) (beatCounter / metre + 1
                    + phrase.getStartTime()), totalBeatWidth - 4,
                    bPos - 50);
                }
                totalBeatWidth += 12;
            }
        }
    }

    // draw treble stave
    for (int i = 0; i < 5; i++) {
        g.drawLine(rightMargin, (bPos + imageHeightOffset - (2 *
        staveSpaceHeight)) + (i * staveSpaceHeight), totalBeatWidth,
        (bPos + imageHeightOffset - (2 * staveSpaceHeight)) + (i *
        staveSpaceHeight));
    }

    // draw bass stave
    for (int i = 6; i < 11; i++) {
        g.drawLine(rightMargin, (bPos + imageHeightOffset - (2 *
        staveSpaceHeight)) + (i * staveSpaceHeight), totalBeatWidth,
        (bPos + imageHeightOffset - (2 * staveSpaceHeight)) + (i *
        staveSpaceHeight));
    }

    g.setColor(Color.darkGray);




    // draw upper treble stave
    for (int i = -7; i < -2; i++) {
        g.drawLine(rightMargin, (bPos + imageHeightOffset - (2 *
        staveSpaceHeight)) + (i * staveSpaceHeight), totalBeatWidth,
        (bPos + imageHeightOffset - (2 * staveSpaceHeight)) + (i *
        staveSpaceHeight));
    }

    // draw lower bass stave
    for (int i = 13; i < 18; i++) {
        g.drawLine(rightMargin, (bPos + imageHeightOffset - (2 *
        staveSpaceHeight)) + (i * staveSpaceHeight), totalBeatWidth,
        (bPos + imageHeightOffset - (2 * staveSpaceHeight)) + (i *
        staveSpaceHeight));
```

```
        }
        // draw next note stave area
        // draw stave
        g.setColor(Color.lightGray);
        for (int i = 0; i < 5; i++) {
            g.drawLine(totalBeatWidth,
            (bPos + imageHeightOffset - (2 * staveSpaceHeight)) + (i *
            staveSpaceHeight),totalBeatWidth + 50,(bPos +
            imageHeightOffset - (2 * staveSpaceHeight)) + (i *
            staveSpaceHeight));
        }
        for (int i = 6; i < 11; i++) {
            g.drawLine(totalBeatWidth,(bPos + imageHeightOffset - (2 *
            staveSpaceHeight)) + (i * staveSpaceHeight),
            totalBeatWidth + 50,(bPos + imageHeightOffset - (2 *
            staveSpaceHeight)) + (i * staveSpaceHeight));
        }
       g.setColor(Color.black);
       // add Clefs
       g.drawImage(trebleClef, rightMargin + 7, bPos - 4, this);
       g.drawImage(bassClef, rightMargin + 7, bPos + staveSpaceHeight *
       6, this);

        /* Draw completed buffer to g */
        graphics.drawImage(image, 0, 0, null);
        // clear image
        // clear
        g.setColor(this.getBackground());
        g.fillRect(0, 0, getSize().width, getSize().height);
        g.setColor(this.getForeground());
    //repaint();
    //g.dispose();
    }
}
```

```
================================================================================
================================================================================
```

```
package diamouses.ui.scoreEditor.staves;

import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.awt.*;
import java.util.Vector;

import javax.swing.ImageIcon;
import javax.swing.JPanel;

import jm.JMC;
import jm.gui.cpn.Images;
import jm.gui.cpn.ToolkitImages;
import jm.music.data.*;
```

```
/**
 * An AWT Component for displaying a Common Practice Notation stave.
 *
 * @author Andrew Brown, Adam Kirby
 * @version 8th July 2001
 */
public abstract class DStave extends JPanel implements JMC,
KeyListener {

    private int selectedNote = -1;
    private int selectedArea = 0;
    protected boolean requiresMoreThanOneImage = false;
    protected double excessRhythmValue = 0.0;
    protected boolean isUp = true;
    protected boolean isNote = false;
    // for double buffering
    public Image image;
    protected Graphics g;
    // attributes
     protected Image trebleClef, bassClef, crotchetUp, crotchetDown,
     quaverDown, quaverUp,semiquaverDown, semiquaverUp, minimDown,
     minimUp, semibreve, dot,semiquaverRest, quaverRest, crotchetRest,
     minimRest, semibreveRest,sharp, flat, natural, one, two, three,
     four, five, six, seven, eight, nine, delete, tieOver, tieUnder;
     public int staveSpaceHeight = 8, rightMargin = 20, beatWidth = 43,
     staveWidth = beatWidth*15,imageHeightOffset = 28, clefWidth = 38,
     timeSigWidth = 5, keySigWidth = 5;
    public int bPos = 28;
    protected Phrase phrase;
    protected Image currImage;
    protected int currBeatWidth, totalBeatWidth;
    protected boolean dottedNote = false;
    protected int[] notePosOffset = {24,24,20,20,16,12,12,8,8,4,4,0}; //
    chromatic scale
    protected double metre = 4.0;
    protected int keySignature = 0; // postive numbers = sharps, negative
    numbers = flats
    protected int[] sharps = {77, 72, 79, 74, 69, 76, 71};
    protected int[] flats = {71, 76, 69, 74, 67, 72, 65};
    protected Vector previouslyChromatic = new Vector();
    protected int[] lineNotes = {0, 1, 4, 7, 8, 11, 14, 15, 17, 18, 21,
    22};
    public Vector notePositions = new Vector();
    protected int maxPitch = 127, minPitch = 0;
    protected String title;
    protected boolean barNumbers = false, editable = true, qtOn = false;
    protected int panelHeight = 110, staveDelta = 0;
    protected boolean displayTitle = false;
    protected Font font = new Font("Helvetica", Font.PLAIN, 10);
    protected int spacingValue = 70;
    private double noteDuration=1.0;
```

```
// constructor

 /**
 * Constructs a new stave to display a blank Phrase using the default
 stave
 * images.
 */
public DStave() {
    this(new Phrase(), new ToolkitImages());
}

 /**
 * Constructs a new stave to display the specified
 <code>phrase</code> using
 * the default stave images.
 *
 * @param phrase    Phrase to be displayed in stave
 */
public DStave(Phrase phrase) {
    this(phrase, new ToolkitImages());
}

public int getSelectedNote(){
    return selectedNote;
}
public void setSelectedNote(int note){
  System.out.println("Selected note = " + note);
    this.selectedNote=note;
}
public int getSelectedArea(){
    return selectedArea;
}
public void setSelectedArea(int area){
    this.selectedArea=area;
}
 /**
 * Constructs a new stave to display a blank Phrase using the
 specified
 * stave <code>images</code>.
 *
 * @param images    Images representing notes, rest and other stave
 elements to use within the compenent
 */

public DStave(Images images) {
    this(new Phrase(), images);
}
 /**
 * Constructs a new stave to display the specified
 <code>phrase</code> using
 * the specified stave <code>images</code>.
 *
 * @param phrase    Phrase to be displayed in stave
 * @param images    Images representing notes, rest and other stave
 elements to use within the compenent
 */
```

```
    public DStave(Phrase phr, Images images) {
        super();
        title = phr.getTitle();
        this.phrase = addRequiredRests(phr);
        // change 'paper' colour

       this.setBackground(Color.getHSBColor((float)0.14,(float)0.09,(flo
       at)1.0)); // .17, .1, 1

      // set the appropriate size (at least 8 bars of 4/4) for the stave
        this.setSize((int)(beatWidth*spacingValue), panelHeight);
if (this.getSize().width < (int)(phrase.getEndTime()* beatWidth * 1.5) )
 this.setSize( (int)(phrase.getEndTime()* beatWidth * 1.5), panelHeight);

        // compensate for overly large images - pain!!
        //if (this.getSize().width > 5000) {
        // this.setSize(5000, panelHeight);
        // System.out.println("Not all the phrase can be shown due to
        overly large image requirements - sorry");
        //}
         //System.out.println("Max size is "+this.getMaximumSize().width
        +" "+ this.getMaximumSize().height);

         // register the listerners
        DStaveActionHandler handleActions = new
        DStaveActionHandler(this);
        this.addMouseListener(handleActions);
        this.addMouseMotionListener(handleActions);
        //  this.addKeyListener(handleActions);


        trebleClef = images.getTrebleClef();
        bassClef = images.getBassClef();
        crotchetDown = images.getCrotchetDown();
        crotchetUp = images.getCrotchetUp();
        quaverDown = images.getQuaverDown();
        quaverUp = images.getQuaverUp();
        semiquaverDown = images.getSemiquaverDown();
        semiquaverUp = images.getSemiquaverUp();
        minimDown = images.getMinimDown();
        minimUp = images.getMinimUp();
        semibreve = images.getSemibreve();
        dot = images.getDot();
        semiquaverRest = images.getSemiquaverRest();
        quaverRest = images.getQuaverRest();
        crotchetRest = images.getCrotchetRest();
        minimRest = images.getMinimRest();
        semibreveRest = images.getSemibreveRest();
        sharp = images.getSharp();
        flat = images.getFlat();
        natural = images.getNatural();
        one = images.getOne();
        two = images.getTwo();
        three = images.getThree();
        four = images.getFour();
        five = images.getFive();
```

```
        six = images.getSix();
        seven = images.getSeven();
        eight = images.getEight();
        nine = images.getNine();
        delete = images.getDelete();
        tieOver = images.getTieOver();
        tieUnder = images.getTieUnder();
    }
     /*
     * Puts rests at the start of an phrase that does not
     * start at time 0.0.
     */
    public Phrase addRequiredRests(Phrase phrase) {
        // add rests if required at the start
        if (phrase.getStartTime() > 0.0) {
            Phrase tempPhrase = new Phrase(0.0);
            double remTime = phrase.getStartTime();
            while(remTime >= 4.0) {
                tempPhrase.addNote(REST, 4.0);
                remTime -= 4.0;
            }
            while(remTime >= 1.0) {
                tempPhrase.addNote(REST, 1.0);
                remTime -= 1.0;
            }
            tempPhrase.addNote(REST, remTime);
            jm.music.tools.Mod.append(tempPhrase, phrase);
            phrase = tempPhrase;
        }
        return phrase;
    }

    /**
     * Sets the current Phrase for this Stave instance
     * @param Phrase
     */

    public void setPhrase(Phrase phr) {
        this.phrase = addRequiredRests(phr);
        previouslyChromatic.removeAllElements();
        //setTitle(phr.getTitle());
        repaint();
    }
     /**
     * Returns the current Phrase of this Stave instance
     */
    public Phrase getPhrase() {
        return this.phrase;
    }
     /**
     * Sets the name for this Stave instance
     * @param String Specify the title of the score
     */
    public void setTitle(String title) {
        this.title = title;
        if(this.phrase != null) this.phrase.setTitle(title);
    }
```

```
/**
 * Returns the name for this Stave instance
 * @return String The title of the score
 */
public String getTitle() {
    return title;
}
/**
 * Emptys the name of this Stave instance
 */
public void removeTitle() {
    this.title = null;
}

/**
 * Show the title or not.
 * @param value True or false
 */
public void setDisplayTitle(boolean value) {
    this.displayTitle = value;
    this.repaint();
}

/**
 * Is the title displayed or not.
 * @param value True or false
 */
public boolean getDisplayTitle() {
    return this.displayTitle;
}

/**
 * Return the recommended height for this stave.
 */
public int getPanelHeight() {
    return panelHeight;
}

 /**
 * Sets the current metre for this Stave instance
 * This effects the displayed time signature. 4.0 = 4/4 etc.
 * @param double
 */
public void setMetre(double timeSig) {
    /*
    System.out.print("Time Sig =");
    System.out.println(timeSig);
    System.out.print("Numerator =");
    System.out.println(phrase.getNumerator());
    System.out.print("Denominator =");
    System.out.println(phrase.getDenominator());
     */
    this.metre = timeSig;
}
```

```
/**
 * returns the current metre for this Stave instance as a double
 */
public double getMetre() {
    return this.metre;
}

/**
 * returns the current major key for this Stave instance as a integer
 * 0 is C, 1 is C#/Db major, 2 is D major, etc
 */
public int getMajorKey() {
    int[] keys = {11, 6, 1, 8, 3, 10, 5, 0, 7, 2, 9, 4, 11, 6, 1};
    return keys[keySignature + 7];
}
/**
 * Sets the current key signature for this Stave instance
 * This effects the displayed key signature. 1 = F# etc.
 * 0 is no key signature, + numbers for sharps, - numbers for flats
 * @param int
 */
public void setKeySignature(int key) {
    this.keySignature = key;
}

/**
* returns the current key signature for this Stave instance as a
double
*/
public int getKeySignature() {
    return this.keySignature;
}
/**
 * Decide to show bar numbers or not
 * @param boolean
 */

public void setBarNumbers(boolean show) {
    this.barNumbers = show;
}

/**
 * Decide to allow stave to be editable or not
 * @param boolean
 */
public void setEditable(boolean state) {
    this.editable = state;
}

/**
 * returns the current minimum MIDI pitch number
 */
public int getMinPitch() {
    return this.minPitch;
}
```

```
/**
 * Decide the minimum MIDI pitch number for this stave
 * @param int
 */
public void setMinPitch(int min) {
    this.minPitch = min;
}

/**
 * returns the current maximum MIDI pitch number
 */
public int getMaxPitch() {
    return this.maxPitch;
}
/**
 * Decide the maxinum MIDI pitch number for this stave
 * @param int
 */
public void setMaxPitch(int max) {
    this.maxPitch = max;
}

/**
 * Returns the current next note position in pixels
 */
public int getTotalBeatWidth() {
    return this.totalBeatWidth;
}
/**
 * Sets the current width of the stave in pixels
 * @param int
 */
public void setTotalBeatWidth(int width) {
    this.totalBeatWidth = width;
}

 /**
 * Returns the current state of barNumber showing
 */
public boolean getBarNumbers() {
    return barNumbers;
}
/**
 * Called by outer containers
 */
public Dimension getPreferredSize() {
  return new Dimension( this.getSize().width, this.getSize().height);
}

/**
 * Returns the current state of QuickTime Playback
 */
public boolean getQtOn() {
    return qtOn;
}
```

```
/**
 * Sets the current state of QuickTime
 * @param boolean
 */
public void setQtOn(boolean state) {
    this.qtOn = state;
}

/**
 * Called by stave action on mouseUp
 * Can be overridden by extending classes
 * to add functionality
 */
public void updateChange() {}

// override update for double buffering
public void update(Graphics g) {
    paintComponent(g);
};

public void paintComponent(Graphics graphics) {
    // overridden by each class which extends Stave
}

/**
 * Remove the last note from the phrase
 */
public void deleteLastNote() {
    if(phrase.size() > 0) {
        phrase.removeNote(phrase.size() -1);
        repaint();
        updateChange();
    }
}

protected void chooseImage(int pitch,
        double rhythmValue,
        int upPitch1,
        int downPitch,
        int upPitch2) {
    if (pitch == Note.REST) {
        isNote = false;
        if (rhythmValue <= 0.0) {
            currImage = delete;
            currBeatWidth = (int) (beatWidth * 0.5);
        } else if (rhythmValue <= 0.2501) {
            currImage = semiquaverRest;
            currBeatWidth  = (int) (beatWidth * 0.5);
        } else if (rhythmValue <= 0.501) {
            currImage = quaverRest;
            currBeatWidth  = (int) (beatWidth * (2.0 / 3.0));
        } else if (rhythmValue <= 0.7501) {
            currImage = quaverRest;
            currBeatWidth  = (int) (beatWidth * (2.0 / 3.0));
            dottedNote = true;
        } else if (rhythmValue <= 1.001) {
```

```
            currImage = crotchetRest;
            currBeatWidth  = beatWidth;
        } else if (rhythmValue <= 1.2501) {
            currImage = crotchetRest;
            currBeatWidth  = beatWidth;
            requiresMoreThanOneImage = true;
            excessRhythmValue = rhythmValue - 1.0;
        } else if (rhythmValue <= 1.501) {
            currImage = crotchetRest;
            currBeatWidth  = (int) (beatWidth * 1.5);
            dottedNote = true;
        } else if (rhythmValue <= 1.7501) {
            currImage = crotchetRest;
            currBeatWidth  = (int) (beatWidth * 1.5);
            dottedNote = true;
            requiresMoreThanOneImage = true;
            excessRhythmValue = rhythmValue - 1.5;
        } else if (rhythmValue <= 2.001) {
            currImage = minimRest;
            currBeatWidth  = (int) (beatWidth * 1.7);
        } else if (rhythmValue <= 2.7501) {
            currImage = minimRest;
            currBeatWidth  = (int) (beatWidth * 1.7);
            requiresMoreThanOneImage = true;
            excessRhythmValue = rhythmValue - 2.0;
        } else if (rhythmValue <= 3.001) {
            currImage = minimRest;
            currBeatWidth  = (int) (beatWidth * 1.9);
            dottedNote = true;
        } else if (rhythmValue <= 3.7501) {
            currImage = minimRest;
            currBeatWidth  = (int) (beatWidth * 1.9);
            dottedNote = true;
            requiresMoreThanOneImage = true;
            excessRhythmValue = rhythmValue -  3.0;
        } else if (rhythmValue <= 4.001) {
            currImage = semibreveRest;
            currBeatWidth  = (int) (beatWidth * 0.5);
        } else {
            currImage = semibreveRest;
            currBeatWidth = (int) (beatWidth * 0.5);
            requiresMoreThanOneImage = true;
            excessRhythmValue = rhythmValue - 4.0;
        }
    } else { // a note rather than a rest
        isNote = true;
        if ((pitch < upPitch1 && pitch >= downPitch)
        || pitch < upPitch2 ) { // stems down
            isUp = true;
            if (rhythmValue <= 0.001) {
                currImage = delete;
                currBeatWidth  = (int) (beatWidth * 0.5);
            } else if (rhythmValue <= 0.2501) {
                currImage = semiquaverUp;
                currBeatWidth  = (int) (beatWidth * 0.5);
            } else if (rhythmValue <= 0.501) {
                currImage = quaverUp;
```

```
            currBeatWidth  = (int) (beatWidth * (2.0 / 3.0));
        } else if (rhythmValue <= 0.7501) {
            currImage = quaverUp;
            currBeatWidth  = (int) (beatWidth * 0.67);
            dottedNote = true;
        } else if (rhythmValue <= 1.001) {
            currImage = crotchetUp;
            currBeatWidth  = beatWidth;
        } else if (rhythmValue <= 1.2501) {
            currImage = crotchetUp;
            currBeatWidth  = beatWidth;
            requiresMoreThanOneImage = true;
            excessRhythmValue = rhythmValue - 1.0;
        } else if (rhythmValue <= 1.501) {
            currImage = crotchetUp;
            currBeatWidth  = (int) (beatWidth * 1.5);
            dottedNote = true;
        } else if (rhythmValue <= 1.7501) {
            currImage = crotchetUp;
            currBeatWidth  = (int) (beatWidth * 1.5);
            dottedNote = true;
            requiresMoreThanOneImage = true;
            excessRhythmValue = rhythmValue - 1.5;
        } else if (rhythmValue <= 2.001) {
            currImage = minimUp;
            currBeatWidth  = (int) (beatWidth * 1.7);
        } else if (rhythmValue <= 2.7501) {
            currImage = minimUp;
            currBeatWidth  = (int) (beatWidth * 1.7);
            requiresMoreThanOneImage = true;
            excessRhythmValue = rhythmValue - 2.0;
        } else if (rhythmValue <= 3.001) {
            currImage = minimUp;
            currBeatWidth  = (int) (beatWidth * 1.9);
            dottedNote = true;
        } else if (rhythmValue <= 3.7501) {
            currImage = minimUp;
            currBeatWidth  = (int) (beatWidth * 1.9);
            dottedNote = true;
            requiresMoreThanOneImage = true;
            excessRhythmValue = rhythmValue - 3.0;
        } else if (rhythmValue <= 4.001) {
            currImage = semibreve;
            currBeatWidth = (int) (beatWidth * 2.25);
        } else {
            currImage = semibreve;
            currBeatWidth = (int) (beatWidth * 2.25);
            requiresMoreThanOneImage = true;
            excessRhythmValue = rhythmValue - 4.0;
        }
    } else { // stem down
        isUp = false;
        if (rhythmValue <= 0.001) {
            currImage = delete;
            currBeatWidth = (int) (beatWidth * 0.5);
        } else if (rhythmValue <= 0.2501) {
            currImage = semiquaverDown;
```

```
                currBeatWidth = (int) (beatWidth * 0.5);
            } else if (rhythmValue <= 0.501) {
                currImage = quaverDown;
                currBeatWidth = (int) (beatWidth * (2.0 / 3.0));
            } else if (rhythmValue <= 0.7501) {
                currImage = quaverDown;
                currBeatWidth = (int) (beatWidth * (2.0 / 3.0));
                dottedNote = true;
            } else if (rhythmValue <= 1.001) {
                currImage = crotchetDown;
                currBeatWidth = beatWidth;
            } else if (rhythmValue <= 1.2501) {
                currImage = crotchetDown;
                currBeatWidth = beatWidth;
                requiresMoreThanOneImage = true;
                excessRhythmValue = rhythmValue - 1.0;
            } else if (rhythmValue <= 1.501) {
                currImage = crotchetDown;
                currBeatWidth  = (int) (beatWidth * 1.5);
                dottedNote = true;
            } else if (rhythmValue <= 1.7501) {
                currImage = crotchetDown;
                currBeatWidth  = (int) (beatWidth * 1.5);
                dottedNote = true;
                requiresMoreThanOneImage = true;
                excessRhythmValue = rhythmValue - 1.5;
            } else if (rhythmValue <= 2.001) {
                currImage = minimDown;
                currBeatWidth  = (int) (beatWidth * 1.7);
            } else if (rhythmValue <= 2.7501) {
                currImage = minimDown;
                currBeatWidth  = (int) (beatWidth * 1.7);
                requiresMoreThanOneImage = true;
                excessRhythmValue = rhythmValue - 2.0;
            } else if (rhythmValue <= 3.001) {
                currImage = minimDown;
                currBeatWidth  = (int) (beatWidth * 1.9);
                dottedNote = true;
            } else if (rhythmValue <= 3.7501) {
                currImage = minimDown;
                currBeatWidth  = (int) (beatWidth * 1.9);
                dottedNote = true;
                requiresMoreThanOneImage = true;
                excessRhythmValue = rhythmValue - 3.0;
            } else if (rhythmValue <= 4.001) {
                currImage = semibreve;
                currBeatWidth  = (int) (beatWidth * 2.25);
            } else {
                currImage = semibreve;
                currBeatWidth = (int) (beatWidth * 2.25);
                requiresMoreThanOneImage = true;
                excessRhythmValue = rhythmValue - 4.0;
            }
        }
    }
}
```

```
    public void keyPressed(KeyEvent e) {
    }

    public void keyReleased(KeyEvent e) {
    }

    public void keyTyped(KeyEvent e) {
        System.out.println(e.getKeyChar());
    }

    public double getNoteDuration(){
        return this.noteDuration;
    }
    public void setNoteDuration(double noteDuration){
        this.noteDuration=noteDuration;
    }

 // This method returns a buffered image with the contents of an image
    public static BufferedImage toBufferedImage(Image image) {
        if (image instanceof BufferedImage) {
            return (BufferedImage)image;
        }

        // This code ensures that all the pixels in the image are loaded
        image = new ImageIcon(image).getImage();

       // Determine if the image has transparent pixels; for this
       method's
       // implementation, see e661 Determining If an Image Has
       Transparent Pixels
       // Create a buffered image with a format that's compatible with
       the screen
        BufferedImage bimage = null;
        GraphicsEnvironment ge =
GraphicsEnvironment.getLocalGraphicsEnvironment();
        try {
            // Determine the type of transparency of the new buffered
            image
            //  int transparency = Transparency.OPAQUE;
            //boolean hasAlpha = hasAlpha(image);
            //if (hasAlpha) {
            int transparency = Transparency.BITMASK;
            //}

            // Create the buffered image
            GraphicsDevice gs = ge.getDefaultScreenDevice();
            GraphicsConfiguration gc = gs.getDefaultConfiguration();
            bimage = gc.createCompatibleImage(
               image.getWidth(null), image.getHeight(null),
               transparency);
        } catch (HeadlessException e) {
            // The system does not have a screen
        }

        if (bimage == null) {
            // Create a buffered image using the default color model
            //int type = BufferedImage.TYPE_INT_RGB;
```

```
            //if (hasAlpha) {
                int type = BufferedImage.TYPE_INT_ARGB;
            //}
            bimage = new BufferedImage(image.getWidth(null),
            image.getHeight(null), type);
        }



        // Copy image to buffered image
        Graphics g = bimage.createGraphics();

        // Paint the image onto the buffered image
        g.drawImage(image, 0, 0, null);
        g.dispose();

        return bimage;
    }


    // This method returns an Image object from a buffered image
    public static Image toImage(BufferedImage bufferedImage) {
        return
Toolkit.getDefaultToolkit().createImage(bufferedImage.getSource());
    }


}
```

```
================================================================================
================================================================================
```

```
package diamouses.ui.scoreEditor.staves;

import java.awt.Cursor;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import javax.swing.JMenuItem;
import javax.swing.JPopupMenu;

import diamouses.ui.scoreEditor.Main;
import jm.JMC;
import jm.music.data.Note;
import jm.music.data.Phrase;

public class DStaveActionHandler implements JMC,
MouseListener, MouseMotionListener, ActionListener, KeyListener {
```

```
private DStave theApp;
private int selectedNote = -1;
private boolean topTimeSelected = false, keySelected = false;
private int clickedPosY, clickedPosX, storedPitch = 72;
private double[] rhythmValues = {104.0, 103.0, 102.0, 101.5, 101.0,
 100.75,100.5, 100.25, 0.0, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 3.0,
 4.0};
private boolean  button1Down  = false;

private JPopupMenu  noteContextMenu;

private JMenuItem   editNote,
        repeatNote,
        makeRest,
        deleteNote,

// constructor
DStaveActionHandler(DStave stave) {
    theApp = stave;

    noteContextMenu = new JPopupMenu();

    repeatNote = new JMenuItem("Repeat Note");
    repeatNote.addActionListener(this);
    noteContextMenu.add(repeatNote );

    makeRest = new JMenuItem("Change to Rest");
    makeRest.addActionListener(this);
    noteContextMenu.add(makeRest);

    deleteNote = new JMenuItem("Delete Note");
    deleteNote.addActionListener(this);
    noteContextMenu.add(deleteNote );


    theApp.add(noteContextMenu);
}
boolean inNoteArea( MouseEvent e ) {
    Integer lastX;
    if (theApp.notePositions.size() < 2) {
        lastX =  new Integer(theApp.getTotalBeatWidth() );
    } else {
        lastX =
        (Integer)theApp.notePositions.elementAt(theApp.notePositions.
        size() -2);
    }
    return (e.getX() <= lastX.intValue() + 15) &&
            (e.getX() < theApp.getTotalBeatWidth() + 50);
}

private void searchForSelectedNote(MouseEvent e) {
    Integer tempX;
    Integer tempY;
    for(int i=0;i< theApp.notePositions.size(); i += 2) {
        tempX = (Integer)theApp.notePositions.elementAt(i);
        tempY = (Integer)theApp.notePositions.elementAt(i+1);
        if((e.getX() > tempX.intValue()) &&
```

```
                    (e.getX() < tempX.intValue() + 15) &&
                    ( e.getY() + theApp.staveDelta >
                    tempY.intValue() + 22) &&
                    (e.getY() + theApp.staveDelta <
                    tempY.intValue()+35)){
                selectedNote = i/2;
                clickedPosY = e.getY() + theApp.staveDelta;
                clickedPosX = e.getX();
                // get out of loop ASAP
                i = theApp.notePositions.size();
                System.out.println("LALA");
            }
        }
    }

    // Mouse Listener stubs
    public void mouseClicked(MouseEvent e) {
      System.out.println("Mouse Clicked");
         if ((((e.getModifiers() == InputEvent.BUTTON3_MASK))) &&
         inNoteArea(e)) {
            searchForSelectedNote(e);
            if ((selectedNote >= 0) && (selectedNote <
            theApp.getPhrase().size())) {
                noteContextMenu.show(theApp, e.getX(), e.getY());
            }
         }else if ((((e.getModifiers() == InputEvent.BUTTON1_MASK))) &&
        inNoteArea(e)) {

            searchForSelectedNote(e);
            if ((selectedNote >= 0) && (selectedNote <
            theApp.getPhrase().size())) {
                if(selectedNote==theApp.getSelectedNote()){
                    theApp.setSelectedNote(-1);
                    theApp.setSelectedArea(0);
                }else{
                    theApp.setSelectedNote(selectedNote);
                    theApp.setSelectedArea(0);
                }
            }else{
                theApp.setSelectedNote(-1);
                theApp.setSelectedArea(0);

            }
        }
        // alg0
        System.out.println(theApp.phrase.getSize());
        Main.setEnabledRemove(theApp.phrase.getSize() >= 1);

    }

    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    //mouseMotionListener stubs
    public void mouseMoved(MouseEvent e) {}
```

```
public void mousePressed(MouseEvent e) {
     if (e.isPopupTrigger() || ((e.getModifiers() &
     InputEvent.BUTTON2_MASK) != 0) || (!theApp.editable))
        return;
    button1Down  = true;
    // add a new note if necessary
    // no notes yet?
    if(!inNoteArea(e)) {
        // new note close to mouse click pitch
        int newPitch = 85 - (e.getY() + theApp.staveDelta -
        theApp.bPos)/2;
        // make sure it is not an accidental
        int[] blackNotes = {1,3,6,8,10};
        boolean white = true;
        for(int k=0;k<blackNotes.length;k++) {
            if(newPitch%12 == blackNotes[k])
                newPitch--;
        }
        Note n = new Note(newPitch,theApp.getNoteDuration());
        Phrase phr = theApp.getPhrase();
        phr.addNote(n);
        theApp.repaint();
        // set cursor
        theApp.setCursor(new Cursor(Cursor.HAND_CURSOR));
        // get ready to drag it
        selectedNote = phr.size()-1;
        //theApp.playCurrentNote(selectedNote);
        clickedPosY = e.getY()  + theApp.staveDelta;
        clickedPosX = e.getX();
    } else {
        searchForSelectedNote(e);
        theApp.setCursor(new Cursor(Cursor.MOVE_CURSOR));
        // check for a click on a note  - head?
    }
    if (selectedNote < 0) { // no note clicked on or yet made
        // check which note to insert
        for(int j=0;j< theApp.notePositions.size() - 2; j += 2) {
             Integer tempX =
             (Integer)theApp.notePositions.elementAt(j);
             Integer nextTempX =
             (Integer)theApp.notePositions.elementAt(j+2);
             if(e.getX() > tempX.intValue() + 15 && e.getX() <
             nextTempX.intValue()) {
                 // change cursor
                 theApp.setCursor(new Cursor(Cursor.HAND_CURSOR));
                 // add new note close to mouse clicked pitch
                int newPitch = 85 - (e.getY() + theApp.staveDelta -
                theApp.bPos)/2;
                 // make sure it is not an accidental
                 int[] blackNotes = {1,3,6,8,10};
                 boolean white = true;
                 for(int k=0;k<blackNotes.length;k++) {
                     if(newPitch%12 == blackNotes[k]) newPitch--;
                 }
                  Note n = new Note(newPitch
                  ,theApp.getNoteDuration());
                 Phrase phr = theApp.getPhrase();
```

```
                phr.getNoteList().insertElementAt(n, j/2 + 1);
                theApp.repaint();
                // play and update variables for dragging it
                selectedNote = j/2 + 1;
                clickedPosY = e.getY() + theApp.staveDelta;
                clickedPosX = e.getX();
                // get out of the loop
                j = theApp.notePositions.size();
            }
        }
    }
    // check if the time signature is clicked
    int theSpace = theApp.rightMargin + theApp.clefWidth +
    theApp.keySigWidth;
    if(e.getX() > theSpace && e.getX() < theSpace + 10) {
        theApp.setCursor(new Cursor(Cursor.MOVE_CURSOR));
        topTimeSelected = true;
        clickedPosY = e.getY() + theApp.staveDelta;
        clickedPosX = e.getX();
    }
    // check if the key signature is clicked
    int theClefSpace = theApp.rightMargin + theApp.clefWidth;
    int minKeySpace = 10;
     if (theApp.keySigWidth > minKeySpace) minKeySpace =
     theApp.keySigWidth;
     if(e.getX() > theClefSpace - 10 && e.getX() < theClefSpace +
     minKeySpace) {
        theApp.setCursor(new Cursor(Cursor.MOVE_CURSOR));
        keySelected = true;
        clickedPosY = e.getY() + theApp.staveDelta;
        clickedPosX = e.getX();
    }

    // alg0
    System.out.println(theApp.phrase.getSize());
        Main.setEnabledRemove(theApp.phrase.getSize() >= 1);

}

public void mouseDragged(MouseEvent e) {
    if ((!button1Down) || (!theApp.editable))
        return;
    //theApp.dragNote(e.getX(), e.getY());
    if (selectedNote >= 0) {
        Phrase phr = theApp.getPhrase();
        Note n = phr.getNote(selectedNote);

         // move note down
        if(e.getY() + theApp.staveDelta > clickedPosY + 2 &&
        theApp.getPhrase().getNote(selectedNote).getPitch() != REST)
        {
            n.setPitch(n.getPitch() - 1);
             if (n.getPitch() < theApp.getMinPitch())
             n.setPitch(theApp.getMinPitch());
            // update the current mouse location
            clickedPosY += 2;
            // update the visual display
```

```
    theApp.repaint();

// keep pitch to reinstate it after displaying a rest
    storedPitch = n.getPitch();
}
// move note up
if(e.getY() + theApp.staveDelta < clickedPosY - 2  &&
theApp.getPhrase().getNote(selectedNote).getPitch() != REST)
{
    n.setPitch(n.getPitch() + 1);
     if (n.getPitch() > theApp.getMaxPitch())
     n.setPitch(theApp.getMaxPitch());
    //theApp.playCurrentNote(selectedNote);
    clickedPosY -= 2;
    theApp.repaint();
    storedPitch = n.getPitch();
}
// move note right - increase RV
if(e.getX() > clickedPosX + 6) {
    double tempRV = n.getRhythmValue();
    int tempPitch = n.getPitch();
    // use +100 numbers for rests
    if (tempPitch == REST) tempRV = tempRV + 100;
    int currRVindex = rhythmValues.length;
    // find current rhythm value and update RV
    for(int i=0; i<rhythmValues.length - 1;i++) {
        if(tempRV == rhythmValues[i])
        n.setRhythmValue(rhythmValues[i + 1]);
    }
    clickedPosX = e.getX();


    // update pitch
    if (n.getRhythmValue() > 100.0) {
        n.setPitch(REST);
        n.setRhythmValue(n.getRhythmValue() - 100);
        // update duration value
        n.setDuration(n.getRhythmValue()*0.9);
    } else {
        if (tempPitch == REST) n.setPitch(storedPitch);
    }
    theApp.repaint();
}
// move note left - decrease RV
if(e.getX() < clickedPosX - 6) {
    double tempRV = n.getRhythmValue();
    int tempPitch = n.getPitch();
    // use +100 numbers for rests
    if (tempPitch == REST) tempRV = tempRV + 100;
    // find current rhythm value position in the array
    int currRVindex = 0;
    for(int i=0; i<rhythmValues.length;i++) {
        if(tempRV == rhythmValues[i]) currRVindex = i;
    }
    // update rv
    if (currRVindex > 0) {
        n.setRhythmValue(rhythmValues[currRVindex - 1]);
```

```
                clickedPosX = e.getX();
                // update pitch
                if (n.getRhythmValue() > 100.0) {
                    n.setPitch(REST);
                    n.setRhythmValue(n.getRhythmValue()-100);
                    // update duration value
                    n.setDuration(n.getRhythmValue()*0.9);
                } else {
                    if (tempPitch == REST) n.setPitch(storedPitch);
                }
                theApp.repaint();
            }
        }
    }
    // check for time signature change
    if (topTimeSelected) {
        //increase?
        if(e.getY() + theApp.staveDelta< clickedPosY - 4  ) {
            theApp.setMetre(theApp.getMetre() + 1.0);
            if (theApp.getMetre() > 9.0) theApp.setMetre(9.0);
            if (theApp.getMetre() < 1.0) theApp.setMetre(1.0);
            theApp.getPhrase().setNumerator((new
            Double(Math.round(theApp.getMetre()))).intValue());
            clickedPosY -= 4;
            theApp.repaint();
            theApp.updateChange();
        }
        //decrease
        if(e.getY() + theApp.staveDelta > clickedPosY + 4) {
            theApp.setMetre(theApp.getMetre() - 1.0);
            if (theApp.getMetre() < 1.0) theApp.setMetre(1.0);
            if (theApp.getMetre() > 9.0) theApp.setMetre(9.0);
            theApp.getPhrase().setNumerator((new
            Double(Math.round(theApp.getMetre()))).intValue());
            clickedPosY += 4;
            theApp.repaint();
            theApp.updateChange();
        }
    }
    // check for key signature change
    if (keySelected) {
        //increase?
        if(e.getY() + theApp.staveDelta < clickedPosY - 4) {
            theApp.setKeySignature(theApp.getKeySignature() + 1);
            if (theApp.getKeySignature() > 7)
            theApp.setKeySignature(7);
            clickedPosY -= 4;
            theApp.repaint();
            theApp.updateChange();
        }
        //decrease
        if(e.getY() + theApp.staveDelta > clickedPosY + 4) {
            theApp.setKeySignature(theApp.getKeySignature() - 1);
            if (theApp.getKeySignature() < -7)
            theApp.setKeySignature(-7);
            clickedPosY += 4;
            theApp.repaint();
```

```
                theApp.updateChange();
            }
        }
    }

    public void mouseReleased(MouseEvent e) {
        button1Down = false;
        if(!theApp.editable) return;
        // delete note if necessary
        for(int i=0; i< theApp.getPhrase().getNoteList().size(); i++) {
            if (theApp.getPhrase().getNote(i).getRhythmValue() == 0.0) {
                theApp.getPhrase().getNoteList().removeElementAt(i);
            }
        }
        // unflag any note being selected
        theApp.repaint();
        theApp.updateChange();
        selectedNote = -1;
        topTimeSelected = false;
        keySelected = false;
        theApp.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    }

    public void actionPerformed(ActionEvent e) {
        Phrase phrase   = theApp.getPhrase();
        Note note       = phrase.getNote(selectedNote);
        if (e.getSource() == editNote) {
            JFrame      editorFrame
                    = new JFrame( "Edit this Note");
            editorFrame.setSize( 400, 400);
            editorFrame.setResizable(true);
            DNoteEditor noteEditor
                    = new DNoteEditor(editorFrame);
            noteEditor.editNote(note, 20, 20);
        } else if (e.getSource() == repeatNote) {
            Note newNote  = note.copy();
            phrase.getNoteList().insertElementAt(
                    newNote,
                    selectedNote
                    );
        } else if (e.getSource() == makeRest) {
            note.setFrequency(Note.REST);
        } else if (e.getSource() == deleteNote) {
            phrase.getNoteList().removeElementAt(selectedNote);
        }
        selectedNote = -1;
        theApp.repaint();
    }

 // key listener stubs
    public void keyPressed(KeyEvent e) {}

    public void keyReleased(KeyEvent e) {}

    public void keyTyped(KeyEvent e) {
//        System.out.println(e.getKeyChar());
        if(e.getKeyChar() == '\b') theApp.deleteLastNote();
```

```
    }
}

================================================================================
================================================================================

package diamouses.ui.scoreEditor.staves;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.Vector;

import diamouses.ui.scoreEditor.GuitarNeck;
import jm.JMC;
import jm.gui.cpn.Images;
import jm.music.data.*;

/**
 * Represents a treble clef stave.
 *
 * @author  Andrew Brown, Adam Kirby
 * @version 1.0.1, 8th July 2001
 */
public class DTrebleStave extends DStave implements JMC{

    private static final class Accidental {

        public static final Accidental NONE = new Accidental("none");

        public static final Accidental SHARP = new Accidental("sharp");

        public static final Accidental NATURAL = new
        Accidental("natural");

        public static final Accidental FLAT = new Accidental("flat");

        private String name;

        // Due to a 1.1 compiler bug this constructor cannot be private
        Accidental(String name) {
            this.name = name;
        }

        public String toString() {
            return name;
        }
    }
    /**
    * Defines a type representing the rules and logic of when
    accidentals
    * should be dispalyed against notes on the stave.
    *
    * Note: In jMusic version 1.2 and earlier this inner class was
    previously
    * called AccidentalDisplayStyle.
    */
```

```
public static abstract class Style {

        int[] sharpPitches = {77, 72, 79, 74, 69, 76, 71};

        int[] flatPitches = {71, 76, 69, 74, 67, 72, 65};

        /**
        * Defines the standard style of displaying accidentals in a
        Common
        * Practice Notation stave.
        */
        public static final Style TRADITIONAL = new Trad();

        /**
        * Defines a style unique to jMusic, which displays an accidental
        in
        * all situations where the status (sharp/flat/natural) of a note
        may
        * be unclear.
        *
        * Note: In jMusic version 1.2 and earlier this field was
        previously
        * called SUPERFLUOUS_SHARPS_AND_FLATS.
        */
        public static final Style JMUSIC = new JMusic();

        private static final class Trad extends Style {

            private boolean[] accidentalRequiredByKeySignature =
                    new boolean[12];

            private static final int[] SHARP_ACCIDENTAL_PAIRS =
            { 0, 0, 1, 1, 2, 3, 3, 4, 4, 5, 5, 6 };

            private static final int[] FLAT_ACCIDENTAL_PAIRS =
            { 0, 1, 1, 2, 2, 3, 4, 4, 5, 5, 6, 6 };

            private int[] degreeToAccidentalPair =
                    SHARP_ACCIDENTAL_PAIRS;

            private boolean[] accidentalInEffect =
                    new boolean[7];

            private int keySignature = 0;

            public Trad() {
                super("Traditional style");
                this.initialise(0);
            }

            private void setBooleanArrayToFalse(boolean[] array) {
                for (int i = 0; i < array.length; i++) {
                    array[i] = false;
                }
            }
```

```
    void initialise(final int keySignature) {
        this.keySignature = keySignature;
        if (keySignature < 0) {
            degreeToAccidentalPair = FLAT_ACCIDENTAL_PAIRS;
        } else {
            degreeToAccidentalPair = SHARP_ACCIDENTAL_PAIRS;
        }
        this.setBooleanArrayToFalse(
                accidentalRequiredByKeySignature);
        accidentalRequiredByKeySignature[1] = true;
        accidentalRequiredByKeySignature[3] = true;
        accidentalRequiredByKeySignature[6] = true;
        accidentalRequiredByKeySignature[8] = true;
        accidentalRequiredByKeySignature[10] = true;
        for (int i = 0; i < Math.abs(keySignature); i++) {
            if (keySignature < 0) {
                accidentalRequiredByKeySignature[
                        flatPitches[i] % 12] = true;
                accidentalRequiredByKeySignature[
                        (flatPitches[i] - 1) % 12] = false;
            } else {
                accidentalRequiredByKeySignature[
                        sharpPitches[i] % 12] = true;
                accidentalRequiredByKeySignature[
                        (sharpPitches[i] + 1) % 12] = false;
            }
        }
        this.setBooleanArrayToFalse(accidentalInEffect);
    }

    Accidental selectAccidental(
            final int pitch,
            final double rhythmValue) {
        if (pitch == Note.REST
                || rhythmValue == 0.0) {
            return Accidental.NONE;
        }

        int degree = pitch % 12; // relative to C not tonic

        int accidentalPair = degreeToAccidentalPair[degree];
        if (accidentalRequiredByKeySignature[degree]
                ^ accidentalInEffect[accidentalPair]) {

            accidentalInEffect[accidentalPair] =
                    ! accidentalInEffect[accidentalPair];

if (degree == 1 || degree == 3 || degree == 6
                    || degree == 8 || degree == 10) {
                if (keySignature > -1) {
                    return Accidental.SHARP;
                } else {
                    return Accidental.FLAT;
                }
            } else {
                return Accidental.NATURAL;
            }
```

```
            }
            return Accidental.NONE;
        }

        void processBarLine() {
            this.setBooleanArrayToFalse(accidentalInEffect);
        }
    };

    private static final class JMusic extends Style {

        private Vector chromaticallyAffectedPitches = new Vector();

        /**
         * Key signature encoded as a signed accidental count.  The
         * following conditions apply:
         * <ul>
         *   <li />0 represents no sharps of flats.
         *   <li />positive <i>n</i> represents <i>n</i> sharps
         *   <li />negative <i>n</i> represents <i>n</n> flats
         * </ul>
         */
        private int keySignature;

        // had difficulty finding a better name because I don't
        // really understand what this variable is and does.  It
        // seems to be closely related to previouslyChromatic. It
        // seems to be a count of accidentals, but in all staves of
        // the range of pitches in the MIDI spec.  So a G Major
        // scale would add 1 to the count for the F# in the treble
        // stave, plus 1 for each F# in octaves above and below
        // that.
        // Odd.
        private int keyAccidentals;

        public JMusic() {
            super("JMusic style (with superfluous sharps and "
                    + "flats)");
            this.initialise(0);
        }

        void initialise(final int keySignature) {
            chromaticallyAffectedPitches = new Vector();
            this.keySignature = keySignature;
            keyAccidentals = 0;
            if (keySignature > 0 && keySignature < 8) {
                for (int i = 0; i < keySignature; i++) {
                    int degree = sharpPitches[i] % 12;
                    for (int j = (int)Note.MIN_PITCH;
                    j <= (int)Note.MAX_PITCH;
                    j++) {
                        if ((j % 12) == degree) {
                            chromaticallyAffectedPitches.addElement(
                                    new Integer(j));
                            keyAccidentals++;
                        }
                    }
```

```
            }
        } else if (keySignature < 0 && keySignature > -8) {
            for (int i = 0; i > keySignature; i--) {
                int degree = flatPitches[-i] % 12;
                for (int j = (int)Note.MIN_PITCH;
                j <= (int)Note.MAX_PITCH;
                j++) {
                    if ((j % 12) == degree) {
                        chromaticallyAffectedPitches.addElement(
                                new Integer(j));
                        keyAccidentals++;
                    }
                }
            }
        }
    }

    Accidental selectAccidental(
            final int pitch,
            final double rhythmValue) {
        if (pitch == Note.REST
                || rhythmValue == 0.0) {
            return Accidental.NONE;
        }
        if ((pitch % 12) == 1 || (pitch % 12) == 3
                || (pitch % 12) == 6 || (pitch % 12) == 8
                || (pitch % 12) == 10) {
            if (keySignature > -1) {
                chromaticallyAffectedPitches.addElement(
                        new Integer(pitch - 1));
                return Accidental.SHARP;
            } else {
                chromaticallyAffectedPitches.addElement(
                        new Integer(pitch + 1));
                return Accidental.FLAT;
            }
        } else {
            int size = chromaticallyAffectedPitches.size();
            int temp;
            for(int j = 0; j < size; j++) {
                temp = ((Integer)
                chromaticallyAffectedPitches.elementAt(
                        j)).intValue();
                if (temp == pitch) {
                    if (j > keyAccidentals-1) {
                        chromaticallyAffectedPitches.
                                removeElementAt(j);
                    }
                    return Accidental.NATURAL;
                }
            }
        }
        return Accidental.NONE;
    }
```

```
        void processBarLine() {

    // do nothing
            }
        };

        private String name;

        // Due to a 1.1 compiler bug this constructor cannot be private
        Style(String name) {
            this.name = name;
        }

        public String toString() {
            return name + " of displaying accidentals";
        }

        abstract void initialise(final int keySignature);

        abstract Accidental selectAccidental(
                final int pitch,
                final double rhythmValue);

        abstract void processBarLine();
    }

    private Style style = new Style.JMusic();

    /**
     * Sets the display style of accidentals for this stave.
     *
     * @param ads   Style to be used by this stave.
     */

public void setAccidentalDisplayStyle(Style ads) {
        if (ads == Style.TRADITIONAL) {
            this.style = new Style.Trad();
        } else if (ads == Style.JMUSIC) {
            this.style = new Style.JMusic();
        } else {
            throw new RuntimeException("Unknown Accidental Display
            Style");
        }
    }

    private int tonic = 0;

    protected int[] scale = JMC.MAJOR_SCALE;

    public static final int MAX_HEIGHT = 500;

    public static final int MAX_WIDTH = 2000;
     /**
     * Constructs a new treble stave to display a blank Phrase using the
     default
     * stave images.
     */
```

```
public DTrebleStave() {
    super();
}

/**
 * Constructs a new treble stave to display the specified
 * <code>phrase</code> using the default stave images.
 *
 * @param phrase    Phrase to be displayed in stave
 */
public DTrebleStave(final Phrase phrase) {
    super(phrase);
}

/**
* Constructs a new treble stave to display a blank Phrase using the
* specified stave <code>images</code>.
*
* @param images    Images representing notes, rest and other stave
elements
*                  to use within the compenent
*/
public DTrebleStave(final Images images) {
    super(images);
}

/**
* Constructs a new treble stave to display the specified
* <code>phrase</code> using the specified stave <code>images</code>.
*
* @param phrase    Phrase to be displayed in stave
* @param images    Images representing notes, rest and other stave
elements
*                  to use within the compenent
*/
public DTrebleStave(final Phrase phrase, final Images images) {
    super(phrase, images);
}

private double beatCounter;

public void paintComponent(Graphics graphics) {
  //        if (phrase == null) {
  //            return;
  //        }

    // set up for double buffering
    if(image == null) {
        image = this.createImage(MAX_WIDTH, MAX_HEIGHT);
        g = image.getGraphics();
    }
    // set font
    g.setFont(font);
    // keep track of the rhythmic values for bar lines
    beatCounter = 0.0;
    // reste note position locations
    notePositions.removeAllElements();
```

```
// add a title if set to be visible
if(getDisplayTitle()) g.drawString(title, rightMargin, bPos -
10);
// insert key signature if required
int keyOffset = 0;

style.initialise(keySignature);

// is the key signature using sharps or flats?
if (keySignature > 0 && keySignature < 8) { // sharp
    for(int ks=0;ks<keySignature; ks++) {
        // claulate position
         int keyAccidentalPosition = notePosOffset[
         sharps[ks]%12] + bPos - 4 + (( 5- sharps[ks]/12) * 24) +
         (( 6- sharps[ks]/12) * 4);
        // draw sharp on treble
        g.drawImage(sharp, rightMargin + clefWidth + keyOffset,
         keyAccidentalPosition, this);
        // indent position
        keyOffset += 10;

        keySigWidth = keyOffset;
    }
} else {
    if (keySignature < 0 && keySignature > -8) { // flat
        for(int ks=0;ks< Math.abs(keySignature); ks++) {
            // claulate position
            int keyAccidentalPosition = notePosOffset[
            flats[ks]%12] + bPos - 4 + (( 5- flats[ks]/12) * 24)
            + (( 6- flats[ks]/12) * 4);
            // draw flat
            g.drawImage(flat, rightMargin + clefWidth +
            keyOffset, keyAccidentalPosition, this);
            // indent position
            keyOffset += 10;
        }
    }
}
keySigWidth = keyOffset + 3;

// insert time signature if required
if ( metre != 0.0) {
    Image[] numbers = {one, two, three, four, five, six, seven,
    eight, nine};

    // top number
    g.drawImage(numbers[(int)metre - 1], rightMargin + clefWidth
    + keySigWidth, bPos + 13, this);
    //bottom number
    g.drawImage(four, rightMargin + clefWidth + keySigWidth ,
    bPos + 29, this);
    timeSigWidth = 30;
} else timeSigWidth = 5;
// set indent position for first note
 totalBeatWidth = rightMargin + clefWidth + keySigWidth +
 timeSigWidth;
```

```
//          firstChords =
//                  ChordAnalysis.getFirstPassChords(phrase, 1.0, tonic,
//                                                   scale);
//          secondChords =
//                  ChordAnalysis.getSecondPassChords(phrase, 1.0, tonic,
//                                                    scale);
        lastChordDisplayed = -1;

        // draw notes and rests
        for(int i = 0; i < phrase.size();i++) {
            int notePitchNum = (int)phrase.getNote(i).getPitch();
            // reset pitch for rests

            // position?
            int pitchTempPos;
            if ( notePitchNum == REST ||
            phrase.getNote(i).getRhythmValue() == 0.0) { // rest or
            delete
            pitchTempPos = notePosOffset[71%12] + bPos - 4 + (( 5- 71/12)
            * 24) + (( 6- 71/12) * 4);
            } else {
            pitchTempPos = notePosOffset[notePitchNum%12] + bPos - 4 + ((
            5- notePitchNum/12) * 24) + (( 6- notePitchNum/12) * 4);
            }

            firstAccidentalDisplayed = false;

            semitoneShiftUp = false;
            isTied = false;
            isFirstNoteInTie = true;
            extraImagesUsed = false;
            savedBeatWidth = totalBeatWidth;
            savedBeatWidth2 = 0;
            double rhythmValue = phrase.getNote(i).getRhythmValue();
            double rvToEndOfBar = metre - (beatCounter % metre);

            while (rvToEndOfBar < rhythmValue) {
                isTied = true;
                drawNote(notePitchNum, rvToEndOfBar,
                        pitchTempPos, i);
                rhythmValue -= rvToEndOfBar;
                rvToEndOfBar = metre - (beatCounter % metre);
            }

            drawNote(notePitchNum, rhythmValue, pitchTempPos, i);


        }

        // draw treble stave
        for(int i = 0; i < 5;i++) {
            g.drawLine( rightMargin, (bPos + imageHeightOffset - (2*
            staveSpaceHeight)) +(i* staveSpaceHeight), totalBeatWidth,
            (bPos + imageHeightOffset - (2* staveSpaceHeight)) +(i*
            staveSpaceHeight));
        }
```

```
            // draw neext note stave area
            // draw stave
            g.setColor(Color.lightGray);
            for(int i = 0; i < 5;i++) {
                g.drawLine( totalBeatWidth, (bPos + imageHeightOffset - (2*
                staveSpaceHeight)) +(i* staveSpaceHeight),totalBeatWidth +
                50, (bPos + imageHeightOffset - (2* staveSpaceHeight)) +(i*
                staveSpaceHeight));
            }
            //for(int i = 6; i < 11;i++) {
            //g.drawLine( totalBeatWidth,
            // (bPos + imageHeightOffset - (2* staveSpaceHeight)) +(i*
            staveSpaceHeight),
            //totalBeatWidth + 50,
            // (bPos + imageHeightOffset - (2* staveSpaceHeight)) +(i*
            staveSpaceHeight));
            //        }
            g.setColor(Color.black);
            // add Clefs
            g.drawImage(trebleClef, rightMargin + 7, bPos - 4, this);
            //g.drawImage(bassClef, rightMargin + 7, bPos + staveSpaceHeight
            * 6, this);

            /* Draw completed buffer to g */

            graphics.drawImage(image, 0, 0, null);
            // clear image
            // clear
            g.setColor(this.getBackground());
            g.fillRect(0,0, MAX_WIDTH, MAX_HEIGHT);
            g.setColor(this.getForeground());
            //repaint();
            //g.dispose();
        }

        private boolean isFirstNoteInTie = true;

    //    private boolean isNote = false;

        private boolean firstAccidentalDisplayed = false;

        private boolean isTied = false;

    //    private boolean isUp = true;

        private boolean semitoneShiftUp = false;

        private boolean extraImagesUsed;

    //    private boolean requiresMoreThanOneImage;

    //    private double excessRhythmValue;

        private int savedBeatWidth;

        private int savedBeatWidth2;
```

```
    private int lastChordDisplayed = -1;

    private int lastPosition = 0;

//    private int[] firstChords = new int[0];

//    private int[] secondChords = new int[0];

     private String[] chordStrings = {"I", "II", "III", "IV", "V", "VI",
     "VII","."};

    private void drawNote(int notePitchNum, final double rhythmValue,
            int pitchTempPos, int noOfNote) {
        requiresMoreThanOneImage = false;
        excessRhythmValue = 0.0;

//        if ((beatCounter % 1.0) == 0.0) {
//            int currentBeat = (int) (beatCounter / 1.0);
//            int total = currentBeat - lastChordDisplayed;
//            int remaining = total;
//            while (lastChordDisplayed < currentBeat) {
//                lastChordDisplayed++;
//
//                remaining--;
//
g.drawString(chordStrings[firstChords[lastChordDisplayed]],
//             (int) (totalBeatWidth - ((totalBeatWidth - lastPosition)
//                                                 * (remaining
//                   / (double) total))),
//                       20);
//                int index = secondChords[lastChordDisplayed];
//                String string = chordStrings[index];
////
g.drawString(chordStrings[secondChords[lastChordDisplayed]],
//                g.drawString(string,
//             (int) (totalBeatWidth - ((totalBeatWidth - lastPosition)
//                                                 * (remaining
//                        / (double) total))),
//                        40);
//            }
//            lastPosition = totalBeatWidth;
//        }

        // choose graphic
        chooseImage( notePitchNum, rhythmValue, 71, 0, 71);

        // draw note/rest
        if (currImage != null) {
                  if (noOfNote==getSelectedNote()) {
                        Note selected_note =
                        phrase.getNote(noOfNote);//.getPitch();

      GuitarNeck.getInstance().setNote(selected_note.getPitch());
                        //-System.out.println(selected_note.toString());
                        BufferedImage bi =
                        DStave.toBufferedImage(currImage);
                        int w = bi.getWidth();
```

```
                        int h = bi.getHeight();
                        int pixel;
                        BufferedImage biOut = new BufferedImage(w, h,
                        bi.TYPE_INT_ARGB_PRE);

                        for (int x = 0; x < w; x++) {
                              for (int y = 0; y < h; y++) {
                                    pixel = bi.getRGB(x, y);
                                    if (pixel == (Color.BLACK).getRGB())
{
                                          pixel = (Color.RED).getRGB();
                                    }
                                    biOut.setRGB(x, y, pixel);
                              }
                        }

                        currImage = DStave.toImage(biOut);
                  }
            }
            g.drawImage(currImage, totalBeatWidth, pitchTempPos, this);
            // store position in a vector

            drawNote2(notePitchNum, rhythmValue - excessRhythmValue,
                    pitchTempPos, noOfNote);
            if (requiresMoreThanOneImage) {
                drawNote(notePitchNum, excessRhythmValue,
                        pitchTempPos, noOfNote);
                extraImagesUsed = true;
            }
        }

        private void drawNote2(int pitch, final double rhythmValue,
                int yCoordinate,int noOfNote) {

            // draw accidental

            if (pitch != Note.REST && rhythmValue != 0.0) {
                Accidental accidental =
                        style.selectAccidental(pitch, rhythmValue);
                if (accidental == Accidental.SHARP) {
                    if (! firstAccidentalDisplayed) {
                        displayImage(g, sharp, totalBeatWidth - 9,
                        yCoordinate, noOfNote);
                    }
                    // enter the note made sharp i.e, F for an F#
                } else if (accidental == Accidental.FLAT) {
                    yCoordinate -= 4; // to show the note a semitone higher
                    for flats
                    if (! firstAccidentalDisplayed) {
                        displayImage(g, flat, totalBeatWidth - 9,
                        yCoordinate, noOfNote);
                    }
                    pitch++; // assume it is a semitone higher for
                    legerlines etc...
                    semitoneShiftUp = true;
                } else if (accidental == Accidental.NATURAL) {
                    if (! firstAccidentalDisplayed) {
```

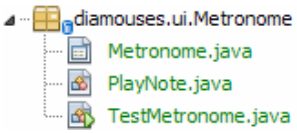
```
                displayImage(g, natural, totalBeatWidth - 7,
                yCoordinate, noOfNote);
            }
        }
    }

    firstAccidentalDisplayed = true;

    // draw note/rest
    displayImage(g, currImage, totalBeatWidth, yCoordinate,
    noOfNote);

    // store position in a vector
    notePositions.addElement(new Integer(totalBeatWidth));
    notePositions.addElement(new Integer(yCoordinate));

    if (dottedNote) {
        boolean dotFlag = true;
        for(int l = 0; l < lineNotes.length; l++) {
            if (lineNotes[l] + 12 == pitch
                    || lineNotes[l] + 36 == pitch
                    || lineNotes[l] + 60 == pitch
                    || lineNotes[l] + 84 == pitch
                    || lineNotes[l] + 108 == pitch
                    || pitch == REST) {
                displayImage(g, dot, totalBeatWidth + 1, yCoordinate
                - 4, noOfNote);
                dotFlag = false;
                l = lineNotes.length;
            }
        }
        if (dotFlag) {
            displayImage(g, dot, totalBeatWidth + 1, yCoordinate,
            noOfNote);
        }
    }

    // leger lines down
    if (pitch <= 61 && pitch > -1 && rhythmValue != 0.0) {g.drawLine(
    totalBeatWidth - 3, bPos + 52, totalBeatWidth+ 12, bPos + 52);}
    if ( pitch <= 58 && pitch > -1 && rhythmValue != 0.0)
    {g.drawLine( totalBeatWidth - 3, bPos + 60, totalBeatWidth+ 12,
    bPos + 60);}
    if ( pitch <= 54 && pitch > -1 && rhythmValue != 0.0)
    {g.drawLine( totalBeatWidth - 3, bPos + 68, totalBeatWidth+ 12,
    bPos + 68);}
    if ( pitch <= 51 && pitch > -1 && rhythmValue != 0.0)
    {g.drawLine( totalBeatWidth - 3, bPos + 76, totalBeatWidth+ 12,
    bPos + 76);}
    if ( pitch <= 48 && pitch > -1 && rhythmValue != 0.0)
    {g.drawLine( totalBeatWidth - 3, bPos + 84, totalBeatWidth+ 12,
    bPos + 84);}

    // leger lines up
    if ( pitch >= 81 && pitch < 128 && rhythmValue != 0.0)
    {g.drawLine( totalBeatWidth - 3, bPos + 4, totalBeatWidth+ 12,
    bPos + 4);}
```

```
        if ( pitch >= 84 && pitch < 128 && rhythmValue != 0.0)
        {g.drawLine( totalBeatWidth - 3, bPos - 4, totalBeatWidth+ 12,
        bPos - 4);}
        if ( pitch >= 88 && pitch < 128 && rhythmValue != 0.0)
        {g.drawLine( totalBeatWidth - 3, bPos - 12, totalBeatWidth+ 12,
        bPos - 12);}
        if ( pitch >= 91 && pitch < 128 && rhythmValue != 0.0)
        {g.drawLine( totalBeatWidth - 3, bPos - 20, totalBeatWidth+ 12,
        bPos - 20);}
        if ( pitch >= 95 && pitch < 128 && rhythmValue != 0.0)
        {g.drawLine( totalBeatWidth - 3, bPos - 28, totalBeatWidth+ 12,
        bPos - 28);}

        // increment everything
        savedBeatWidth2 = totalBeatWidth;

        if ((isTied || extraImagesUsed) && isNote && ! isFirstNoteInTie)
{

            int yPosition = yCoordinate + 19 - ((semitoneShiftUp) ? 4 :
            0);

            if (isUp) {
                g.drawImage(tieUnder,
                        savedBeatWidth - 3 + 9,
                        yPosition + 17,
                        savedBeatWidth2 + 19 - 9,
                        yPosition + 17 + tieUnder.getHeight(this),
                        0, 0, tieUnder.getWidth(this),
                        tieUnder.getHeight(this),this);
            } else {
                g.drawImage(tieOver,
                        savedBeatWidth - 3 + 9,
                        yPosition - 20,
                        savedBeatWidth2 + 19 - 9,
                        yPosition - 20 + tieOver.getHeight(this),
                        0, 0, tieOver.getWidth(this),
                        tieOver.getHeight(this),
                        this);
            }
        }

        if (isFirstNoteInTie = true) {
            isFirstNoteInTie = false;
        }

        savedBeatWidth = totalBeatWidth;

        totalBeatWidth += currBeatWidth;
        dottedNote = false;
        // quantised to semiquvers!
        // (int)((rhythmValue/0.25) * 0.25);
        beatCounter += (int)(rhythmValue/0.25) * 0.25;

        // draw bar line
        if (metre != 0.0) {
            if ( (beatCounter % metre) == 0.0) {
```

```
                    g.drawLine( totalBeatWidth , bPos + 12, totalBeatWidth,
                    bPos + 44);
                    style.processBarLine();
                    // add bar numbers?
                    if (barNumbers) g.drawString( ""+(int)(beatCounter/metre
                    +1 + phrase.getStartTime()) , totalBeatWidth - 4 ,
                    bPos);
                    totalBeatWidth += 12;
                }
            }


        }

        private void displayImage(final Graphics g, final Image image, int
        xCoord,int yCoord, int noOfNote ) {
            g.drawImage(image, xCoord, yCoord, this);
        }
}
```

==========

==========




```
package diamouses.ui.Metronome;
/*
 * PlayNote.java
 *
 * Created on June 12, 2007, 3:05 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

import java.util.TimerTask;
import javax.sound.midi.Instrument;
import javax.sound.midi.MidiChannel;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.Patch;
import javax.sound.midi.Soundbank;
import javax.sound.midi.Synthesizer;
/** Used by MetronomeDialog to play a midi note.
 *
 * @author Aaron Bauer
 */
public class PlayNote extends TimerTask{
    private MidiChannel[] channels;
    private int beatCount;
    private static int noteCount;
    private boolean compoundTime;
```

```
    /** Creates a new instance of PlayNote.  Requires beats per measure
    integer
    *  and a compound time boolean
    * @param beatsPM - The beats per minute
    * @param compound - whether the compound time is used
    */
    public PlayNote(int beatsPM, boolean compound) {
        PlayNote.noteCount = beatsPM;
        this.beatCount = beatsPM - 1;
        this.compoundTime = compound;
        try
        {
        //initializes midi
            Synthesizer synth = MidiSystem.getSynthesizer();
            synth.open();
             this.channels = synth.getChannels();
            Soundbank bank = synth.getDefaultSoundbank();
            synth.loadAllInstruments(bank);
            Instrument instrs[] = synth.getLoadedInstruments();
            Instrument instr = instrs[115];
            Patch instrPatch = instr.getPatch();
            this.channels[2].programChange(instrPatch.getBank(),
                instrPatch.getProgram());
        }
        catch (Exception exc){
            exc.printStackTrace();
        }
    }
/** Plays the note depending on beats per measure and whether or not
compound
 * time is enabled.
 */
    public void run() {
        this.beatCount = this.beatCount + 1;
        if(this.beatCount == PlayNote.noteCount){
            this.channels[2].noteOn(60, 70);
            try {
                  Thread.sleep(1);
            }
            catch (Exception exc){
            exc.printStackTrace();
            }
            this.channels[2].noteOff(60);
            this.beatCount = 0;
        }
         else if(this.compoundTime && (this.beatCount == 3 ||
         this.beatCount == 6 || this.beatCount == 9) ){
            this.channels[2].noteOn(45, 70);
            try {
                  Thread.sleep(1);
            }
            catch (Exception exc){
            exc.printStackTrace();
            }
            this.channels[2].noteOff(45);
        }
```

```
        else {
            this.channels[2].noteOn(50, 70);
            try {
                  Thread.sleep(1);
            }
            catch (Exception exc){
            exc.printStackTrace();
            }
            this.channels[2].noteOff(50);
        }
    }

}
```

```
================================================================================
================================================================================
```

```
package diamouses.ui.Metronome;

import javax.swing.JFrame;
public class TestMetronome {

      public static void main (String [] args) {
        Metronome met = new Metronome(new JFrame(), null, true);
        met.setVisible(true);
       }
}
```

```
================================================================================
================================================================================
```

```
package diamouses.ui.Metronome;

import java.awt.*;
import javax.swing.*;
import java.util.Timer;

/**  Creates a metronome that works by having a timer play a note at a
 * user defined interval in beats per minute.
 *
 * @author  Aaron Bauer
 */
public class Metronome extends javax.swing.JDialog
{

    PlayNote playNote = null;
    // beats per minute
    private int bpm = 120;
    private Timer noteTimer = new Timer();
    // beats per measure
    private double beats = 4;
    // compound time indicator
    private boolean compound = false;
    // used to check text field content
```

```
    private double textTest;

    /** Creates new form Metronome
     * @param parent - The parent frame
     * @param image - The title image
     * @param modal - The modality of the screen
     */
    public Metronome(java.awt.Frame parent, Image image, boolean modal)
    {
        super(parent, modal);
        this.setIconImage(image);
        initComponents();
        this.rb44.setSelected(true);
        noteTimer = new Timer();
    }


    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc="Generated
    Code">//GEN-BEGIN:initComponents
    private void initComponents() {

        buttonGroup1 = new javax.swing.ButtonGroup();
        rb24 = new javax.swing.JRadioButton();
        jLabel4 = new javax.swing.JLabel();
        rb68 = new javax.swing.JRadioButton();
        rb98 = new javax.swing.JRadioButton();
        rb128 = new javax.swing.JRadioButton();
        rbCustom = new javax.swing.JRadioButton();
        jLabel3 = new javax.swing.JLabel();
        rb44 = new javax.swing.JRadioButton();
        rb54 = new javax.swing.JRadioButton();
        rb34 = new javax.swing.JRadioButton();
        jTextField2 = new javax.swing.JTextField();
        jLabel6 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jLabel1 = new javax.swing.JLabel();
        jSlider1 = new javax.swing.JSlider();
        btnStart = new javax.swing.JButton();
        jLabel2 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();


setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Score-Editor Metronome");
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                formWindowClosing(evt);
            }
        });

        buttonGroup1.add(rb24);
```

```
rb24.setText("2/4");
rb24.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0,
0, 0));
 rb24.setMargin(new java.awt.Insets(0, 0, 0, 0));
 rb24.addActionListener(new java.awt.event.ActionListener() {
     public void actionPerformed(java.awt.event.ActionEvent evt) {
         rb24ActionPerformed(evt);
     }
 });

 jLabel4.setText("Compound Time");

 buttonGroup1.add(rb68);
 rb68.setText("6/8");
rb68.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0,
0, 0));
 rb68.setMargin(new java.awt.Insets(0, 0, 0, 0));
 rb68.addActionListener(new java.awt.event.ActionListener() {
     public void actionPerformed(java.awt.event.ActionEvent evt) {
         rb68ActionPerformed(evt);
     }
 });

 buttonGroup1.add(rb98);
 rb98.setText("9/8");
rb98.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0,
0, 0));
 rb98.setMargin(new java.awt.Insets(0, 0, 0, 0));

 buttonGroup1.add(rb128);
 rb128.setText("12/8");
rb128.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0,
0, 0));
 rb128.setMargin(new java.awt.Insets(0, 0, 0, 0));
 rb128.addActionListener(new java.awt.event.ActionListener() {
     public void actionPerformed(java.awt.event.ActionEvent evt) {
         rb128ActionPerformed(evt);
     }
 });

 buttonGroup1.add(rbCustom);
 rbCustom.setText("Custom Time");
rbCustom.setBorder(javax.swing.BorderFactory.createEmptyBorder(0,
0, 0, 0));
 rbCustom.setMargin(new java.awt.Insets(0, 0, 0, 0));
 rbCustom.addActionListener(new java.awt.event.ActionListener() {
     public void actionPerformed(java.awt.event.ActionEvent evt) {
         rbCustomActionPerformed(evt);
     }
 });

 jLabel3.setText("Simple Time");

 buttonGroup1.add(rb44);
 rb44.setText("4/4");
rb44.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0,
0, 0));
```

```
        rb44.setMargin(new java.awt.Insets(0, 0, 0, 0));
        rb44.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                rb44ActionPerformed(evt);
            }
        });

        buttonGroup1.add(rb54);
        rb54.setText("5/4");
        rb54.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0,
        0, 0));
        rb54.setMargin(new java.awt.Insets(0, 0, 0, 0));
        rb54.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                rb54ActionPerformed(evt);
            }
        });

        buttonGroup1.add(rb34);
        rb34.setText("3/4");
        rb34.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0,
        0, 0));
        rb34.setMargin(new java.awt.Insets(0, 0, 0, 0));
        rb34.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                rb34ActionPerformed(evt);
            }
        });

        jTextField2.setText("4");

        jLabel6.setText("Beats Per Measure");

        jLabel5.setText("Allegro");

        jLabel1.setFont(new java.awt.Font("Tahoma", 0, 12));
        jLabel1.setText("BPM: 120");

        jSlider1.setMaximum(208);
        jSlider1.setMinimum(40);
        jSlider1.setPaintTicks(true);
        jSlider1.setSnapToTicks(true);
        jSlider1.setValue(120);
        jSlider1.addChangeListener(new javax.swing.event.ChangeListener()
{
            public void stateChanged(javax.swing.event.ChangeEvent evt) {
                jSlider1StateChanged(evt);
            }
        });

        btnStart.setText("Start");
        btnStart.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnStartActionPerformed(evt);
            }
        });
```

```
        jLabel2.setFont(new java.awt.Font("Tahoma", 1, 18));
        jLabel2.setText("Metronome");

        javax.swing.GroupLayout layout = new
        javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                    .addComponent(rb98)
                    .addComponent(rb68)
                    .addComponent(jLabel4)
                    .addComponent(rb24)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                            .addComponent(rbCustom)
                            .addComponent(rb54)
                            .addComponent(rb44)
                            .addComponent(jLabel3)
                            .addComponent(rb34))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                            .addGroup(layout.createSequentialGroup()
                                .addGap(24, 24, 24)
                                .addComponent(jLabel6))
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jSlider1,
javax.swing.GroupLayout.PREFERRED_SIZE, 247,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(btnStart))
                            .addGroup(layout.createSequentialGroup()
                                .addGap(91, 91, 91)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                                    .addComponent(jLabel5,
javax.swing.GroupLayout.PREFERRED_SIZE, 69,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jLabel1)))
                            .addGroup(layout.createSequentialGroup()
                                .addGap(70, 70, 70)
                                .addComponent(jLabel2))
                            .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE, 20,
javax.swing.GroupLayout.PREFERRED_SIZE)))
```

```
                    .addComponent(rb128))
                .addContainerGap(12, Short.MAX_VALUE))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(203, Short.MAX_VALUE)
                .addComponent(jLabel7)
                .addGap(214, 214, 214))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(28, 28, 28)
                        .addComponent(jLabel3)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(rb44)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(rb54)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(rb34)
                        .addGap(6, 6, 6)
                        .addComponent(rb24)
                        .addGap(6, 6, 6)
                        .addComponent(jLabel4)
                        .addGap(6, 6, 6)
                        .addComponent(rb68)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(rb98))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jLabel2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TR
AILING)
                            .addComponent(btnStart)
                            .addComponent(jSlider1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jLabel1)
                        .addGap(6, 6, 6)
                        .addComponent(jLabel5)))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(rb128)
```

```
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                    .addComponent(rbCustom)
                    .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel6))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 20,
Short.MAX_VALUE)
                .addComponent(jLabel7)
                .addContainerGap())
        );

        java.awt.Dimension screenSize =
java.awt.Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((screenSize.width-425)/2, (screenSize.height-300)/2,
425, 300);
    }// </editor-fold>//GEN-END:initComponents

private void formWindowClosing(java.awt.event.WindowEvent evt) {//GEN-
FIRST:event_formWindowClosing

    noteTimer.purge();
    noteTimer.cancel();
}//GEN-LAST:event_formWindowClosing

private void jSlider1StateChanged(javax.swing.event.ChangeEvent evt)
{//GEN-FIRST:event_jSlider1StateChanged
    JSlider source = (JSlider) evt.getSource();
    if(!source.getValueIsAdjusting())
    {
        bpm = source.getValue();
        jLabel1.setText("BPM: " + bpm);
        if(bpm <= 59 && bpm >= 40)
        {
            jLabel5.setText("Largo");
        }
        if(bpm <= 65 && bpm >= 60)
        {
            jLabel5.setText("Larghetto");
        }
        if(bpm <= 75 && bpm >= 66)
        {
            jLabel5.setText("Adagio");
        }
        if(bpm <= 107 && bpm >= 76)
        {
            jLabel5.setText("Andante");
        }
        if(bpm <= 119 && bpm >= 108)
        {
            jLabel5.setText("Moderato");
```

```
            }
            if(bpm <= 167 && bpm >= 120)
            {
               jLabel5.setText("Allegro");
            }
            if(bpm <= 199 && bpm >= 168)
            {
                jLabel5.setText("Presto");
            }
            if(bpm <= 208 && bpm >= 200)
            {
                jLabel5.setText("Prestissimo");
            }
            if(btnStart.getText().equals("Stop"))
            {
                double noteLength = 1 / (((double) this.bpm / 60) / 1000);
                playNote.cancel();
                playNote = new PlayNote((int) this.beats, this.compound);
                noteTimer.schedule(playNote, 0, (long) noteLength);
            }
        }
    }//GEN-LAST:event_jSlider1StateChanged

    private void btnStartActionPerformed(java.awt.event.ActionEvent evt)
    {//GEN-FIRST:event_btnStartActionPerformed
         if(btnStart.getText().equals("Start"))
            {
                btnStart.setText("Stop");
                double noteLength = 1 / (((double) bpm / 60) / 1000);
                playNote = new PlayNote((int) this.beats, this.compound);
                noteTimer.schedule(playNote, 0, (long) noteLength);
            }
            else
            {
                noteTimer.purge();
                playNote.cancel();
                btnStart.setText("Start");
            }
    }//GEN-LAST:event_btnStartActionPerformed

    private void rb44ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
    FIRST:event_rb44ActionPerformed
        resetTempo();
    }//GEN-LAST:event_rb44ActionPerformed

    private void rb54ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
    FIRST:event_rb54ActionPerformed
        resetTempo();
    }//GEN-LAST:event_rb54ActionPerformed

    private void rb34ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
    FIRST:event_rb34ActionPerformed
        resetTempo();
    }//GEN-LAST:event_rb34ActionPerformed

    private void rb24ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
    FIRST:event_rb24ActionPerformed
```

```
    resetTempo();
}//GEN-LAST:event_rb24ActionPerformed

private void rb68ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_rb68ActionPerformed
    resetTempo();
}//GEN-LAST:event_rb68ActionPerformed

private void rb128ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_rb128ActionPerformed
    resetTempo();
}//GEN-LAST:event_rb128ActionPerformed

private void rbCustomActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_rbCustomActionPerformed
    resetTempo();
}//GEN-LAST:event_rbCustomActionPerformed

    private void resetTempo()
    {
        if(btnStart.getText().equals("Stop"))
        {
        //    btnStart.setText("Stop");
            double noteLength = 1 / (((double) bpm / 60) / 1000);
            if(this.rb44.isSelected())
            {
                beats = 4;
                compound = false;
            }
            if(rbCustom.isSelected())
            {
                // allows program to handle illogical entry into text
                field

            try
                {
                    textTest =
Double.valueOf(this.jTextField2.getText());
                }
                catch(NumberFormatException ex)
                {
                    ex.printStackTrace();
                }
                if(this.textTest >= 1 && this.textTest < 101)
                {
                    beats = Double.valueOf(this.jTextField2.getText());
                    compound = false;
                    jLabel7.setText("");
                }
                else
                {
                    jLabel7.setText("Try a different custom time.");
                }
            }
            if(this.rb54.isSelected())
            {
                beats = 5;
```

```
            compound = false;
        }
        if(rb34.isSelected())
        {
            beats = 3;
            compound = false;
        }
        if(rb24.isSelected())
        {
            beats = 2;
            compound = false;
        }
        if(rb68.isSelected())
        {
            beats = 6;
            compound = true;
        }
        if(rb98.isSelected())
        {
            beats = 9;
            compound = true;
        }
        if(rb128.isSelected())
        {
            beats = 12;
            compound = true;
        }
            noteTimer.purge();
            if(playNote != null)
            {
                playNote.cancel();
                playNote = null;
            }
            playNote = new PlayNote((int) this.beats, this.compound);
            noteTimer.schedule(playNote, 0, (long) noteLength);
        //}
    }
    else
    {
        noteTimer.purge();
        playNote.cancel();
    }
}

// Variables declaration - do not modify//GEN-BEGIN:variables
private javax.swing.JButton btnStart;
private javax.swing.ButtonGroup buttonGroup1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JSlider jSlider1;
private javax.swing.JTextField jTextField2;
private javax.swing.JRadioButton rb128;
```

```
    private javax.swing.JRadioButton rb24;
    private javax.swing.JRadioButton rb34;
    private javax.swing.JRadioButton rb44;
    private javax.swing.JRadioButton rb54;
    private javax.swing.JRadioButton rb68;
    private javax.swing.JRadioButton rb98;
    private javax.swing.JRadioButton rbCustom;
    // End of variables declaration//GEN-END:variables
}
```

```
<?xml version="1.0"?>
<!-- ======================================================================= -->
<!-- A Buildfile                                                             -->
<!-- ======================================================================= -->
<project name="ScoreEditor" default="jar" basedir=".">

 <target name="variables">
   <tstamp/>

   <property name="Name"              value="ScoreEditor"/>

   <property name="src.dir" value="${basedir}${file.separator}src"/>

   <property name="lib.dir"          value="${basedir}${file.separator}libs"/>
   <property name="build.dir"        value="${basedir}${file.separator}build"/>
   <property name="build.classes"value="${build.dir}${file.separator}classes"/>
   <property name="build.jar.dir" value="${build.dir}${file.separator}jars"/>
   <property name="build.jar"
value="${build.jar.dir}${file.separator}${Name}.jar"/>

<property name="build.classpath"
value=".:${lib.dir}${file.separator}jmusic.jar"/>

 <property name="images.src"
value="${src.dir}${file.separator}diamouses${file.separator}ui${file.sepa
rator}scoreEditor${file.separator}images"/>

<property name="images.dest"
value="${build.classes}${file.separator}diamouses${file.separator}ui${fil
e.separator}scoreEditor${file.separator}images"/>

   <property name="packages"   value="*"/>
</target>
<!-- ======================================================================= -->
<!-- Create required directories                                             -->
<!-- ======================================================================= -->
    <target name="init" depends="variables">
        <!-- Prepare necessary directories -->
        <mkdir dir="${build.dir}"/>
        <mkdir dir="${build.classes}"/>
        <mkdir dir="${build.jar.dir}"/>
        <mkdir dir="${images.dest}"/>

        <!-- Copy all the images to the build directory -->
        <copy todir="${images.dest}">
            <fileset dir="${images.src}" casesensitive="yes">
                <include name="**/*.*"/>
            </fileset>
        </copy>
</target>
```

```
<!-- ================================================================== -->
<!-- Compiles the source code                                           -->
<!-- ================================================================== -->
    <target name="compile" depends="init" description="Compiles source
code">
        <javac encoding="UTF-8"
               srcdir="${src.dir}"
               destdir="${build.classes}"
               classpath="${build.classpath}"
               optimize="on"
                />
    </target>
<!-- ================================================================== -->
<!-- Creates a jar archive                                              -->
<!-- ================================================================== -->
    <target name="jar" depends="init,compile" description="Generates
ScoreEditor.jar" >
<!-- Put everything from ${build.dir} into the ScoreEditor.jar file -->
        <jar
                manifest="${src.dir}/myManifest.txt"
                jarfile="${build.jar.dir}${file.separator}${Name}.jar"
                basedir="${build.classes}"
                includes="**"
                />
        <copy todir="${build.jar.dir}">
            <fileset dir="${src.dir}" casesensitive="yes">
                <include name="**/*.html"/>
            </fileset>
        </copy>
        <copy todir="${build.jar.dir}">
            <fileset dir="${lib.dir}" casesensitive="yes">
                <include name="**/*.jar"/>
            </fileset>
        </copy>
    </target>
<!-- ================================================================== -->
<!-- Cleans up all                                                      -->
<!-- ================================================================== -->
    <target name="clean" depends="init" description="Removes previous
build (classes and jar files)">
        <delete dir="${build.classes}"/>
        <delete dir="${build.jar.dir}"/>
    </target>
<!-- ================================================================== -->
<!-- Run Score Editor                                                   -->
<!-- ================================================================== -->
    <target name="run" depends="init,compile,jar" description="Initiates
ScoreEditor.java">
        <echo message = "Running ScoreEditor"/>
        <java
                        jar="${build.jar.dir}${file.separator}${Name}.jar"
      fork="true"

        >
        </java>
    </target>
</project>
```

JavaDoc of GuitarNeck object

diamouses.ui.scoreEditor

Class GuitarNeck

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ javax.swing.JComponent
              └ javax.swing.JPanel
                  └ diamouses.ui.scoreEditor.GuitarNeck
```

All Implemented Interfaces:

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

---

```
public class GuitarNeck
extends javax.swing.JPanel
```

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JPanel
javax.swing.JPanel.AccessibleJPanel

Nested classes/interfaces inherited from class javax.swing.JComponent
javax.swing.JComponent.AccessibleJComponent

Nested classes/interfaces inherited from class java.awt.Container
java.awt.Container.AccessibleAWTContainer

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

Field Summary	
(package private) java.awt.Color	**color_1st**
(package private) java.awt.Color	**color_2nd**
(package private) java.awt.Color	**color_3rd**
(package private) java.awt.Color	**color_4th**
(package private) java.awt.Color	**color_5th**
(package private) java.awt.Color	**color_6th**
private javax.swing.JToggleButton[][]	**fret_buttons**
private javax.swing.JPanel	**fret_panel**
private int	**height** Global variables for the application
(package private) javax.swing.ImageIcon	**ii**
private static GuitarNeck	**reference**
private int	**screen_width**
private javax.swing.JPanel[]	**string_panels**

Fields inherited from class javax.swing.JComponent
accessibleContext, listenerList, TOOL_TIP_TEXT_KEY, ui, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW

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

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

Constructor Summary

private	**GuitarNeck**(int screen_width) Constructor.

Method Summary

void	**clearAll**()
private javax.swing.ImageIcon[]	**getIconsForString**(int string) Method getIconsForString returns an Array [] of ImageIcons with the images of the appropriate Notes for each string of the Guitar.
static GuitarNeck	**getInstance**()
static GuitarNeck	**getInstance**(int mm)
private javax.swing.JPanel	**getStringPanel**() Method returns a JPanel with JButtons for each note of the Guitar
static void	**main**(java.lang.String[] args) Main -> For testing purposes
void	**paintComponent**(java.awt.Graphics g) Method Paint the Guitar Neck on the Background of this JPanel.
void	**setNote**(int pitch) Sets the note with a specific pitch on the virtual guitar
private void	**setSizes**() This method sets the sizes of the JToggleButtons
private void	**switchColor**(int string) Switch the color of the string

Methods inherited from class javax.swing.JPanel

getAccessibleContext, getUI, getUIClassID, paramString, setUI, updateUI

Methods inherited from class javax.swing.JComponent

addAncestorListener, addNotify, addVetoableChangeListener, computeVisibleRect, contains, createToolTip, disable, enable, firePropertyChange, firePropertyChange, firePropertyChange, fireVetoableChange, getActionForKeyStroke, getActionMap, getAlignmentX, getAlignmentY, getAncestorListeners, getAutoscrolls, getBaseline, getBaselineResizeBehavior, getBorder, getBounds, getClientProperty, getComponentGraphics, getComponentPopupMenu, getConditionForKeyStroke,

getDebugGraphicsOptions, getDefaultLocale, getFontMetrics, getGraphics, getHeight, getInheritsPopupMenu, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets, getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPopupLocation, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor, getTransferHandler, getVerifyInputWhenFocusTarget, getVetoableChangeListeners, getVisibleRect, getWidth, getX, getY, grabFocus, isDoubleBuffered, isLightweightComponent, isManagingFocus, isOpaque, isOptimizedDrawingEnabled, isPaintingForPrint, isPaintingTile, isRequestFocusEnabled, isValidateRoot, paint, paintBorder, paintChildren, paintImmediately, paintImmediately, print, printAll, printBorder, printChildren, printComponent, processComponentKeyEvent, processKeyBinding, processKeyEvent, processMouseEvent, processMouseMotionEvent, putClientProperty, registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeNotify, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, resetKeyboardActions, reshape, revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setComponentPopupMenu, setDebugGraphicsOptions, setDefaultLocale, setDoubleBuffered, setEnabled, setFocusTraversalKeys, setFont, setForeground, setInheritsPopupMenu, setInputMap, setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setTransferHandler, setUI, setVerifyInputWhenFocusTarget, setVisible, unregisterKeyboardAction, update

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, addImpl, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getLayout, getMousePosition, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, printComponents, processContainerEvent, processEvent, remove, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusCycleRoot, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setLayout, transferFocusBackward, transferFocusDownCycle, validate, validateTree

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, createImage, createImage, createVolatileImage, createVolatileImage, disableEvents, dispatchEvent, enable, enableEvents,

enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getForeground, getGraphicsConfiguration, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocale, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getToolkit, getTreeLock, gotFocus, handleEvent, hasFocus, hide, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processMouseWheelEvent, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setFocusable, setFocusTraversalKeysEnabled, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus, transferFocusUpCycle

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

height

```
private int height
```

Global variables for the application

screen_width

```
private int screen_width
```

fret_buttons

```
private javax.swing.JToggleButton[][] fret_buttons
```

---

string_panels
```
private javax.swing.JPanel[] string_panels
```

fret_panel
```
private javax.swing.JPanel fret_panel
```

ii
```
javax.swing.ImageIcon ii
```

color_1st
```
java.awt.Color color_1st
```

color_2nd
```
java.awt.Color color_2nd
```

color_3rd
```
java.awt.Color color_3rd
```

color_4th
```
java.awt.Color color_4th
```

color_5th
```
java.awt.Color color_5th
```

color_6th
```
java.awt.Color color_6th
```

reference
```
private static GuitarNeck reference
```

Constructor Detail

GuitarNeck

```
private GuitarNeck(int screen_width)
```
Constructor. Initializes 6 JPanels, one for each guitar string

Method Detail

getInstance
```
public static GuitarNeck getInstance(int mm)
```

---

getInstance

`public static GuitarNeck getInstance()`

---

getStringPanel

`private javax.swing.JPanel getStringPanel()`

Method returns a JPanel with JButtons for each note of the Guitar

Parameters:

`string` - First..Sixth string of the guitar

Returns:

a JPanel with appropriate JButtons

---

setSizes

`private void setSizes()`

This method sets the sizes of the JToggleButtons

---

paintComponent

`public void paintComponent(java.awt.Graphics g)`

Method Paint the Guitar Neck on the Background of this JPanel.

Overrides:

`paintComponent` in class `javax.swing.JComponent`

---

switchColor

`private void switchColor(int string)`

Switch the color of the string

Parameters:

`string` - values 1 to 6

---

getIconsForString

`private javax.swing.ImageIcon[] getIconsForString(int string)`

Method getIconsForString returns an Array [] of ImageIcons with the images of the appropriate Notes for each string of the Guitar.

Parameters:

`string` - An integer from 1 to 6 representing the First..Sixth string of a Guitar.

Returns:

an ImageIcon [] array.

---

setNote

`public void setNote(int pitch)`

Sets the note with a specific pitch on the virtual guitar

Parameters:

`pitch` - the pitch value.

clearAll
`public void clearAll()`

main
`public static void main(java.lang.String[] args)`

Main -> For testing purposes

Parameters:

`args` -

Overview **Package** **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES** **All Classes** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD