ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ: "Εφαρμογή Υπολογισμού και Απεικόνισης των Χαρακτηριστικών Σημάτων Φωνής σε Συσκευές Android"

Φοιτητές:

Ιωσηφίδης Γεώργιος

Μπόζιας Κωνσταντίνος

ΕΠΙΒΛΕΠΩΝ: Μαριάτος Ευαγγελινός

ΝΑΥΠΑΚΤΟΣ, ΙΟΥΝΙΟΣ 2014

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Αντίρριο, 13/06/2014

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Υπογραφή

2. Υπογραφή

3. Υπογραφή

ΕΙΣΑΓΩΓΗ

Η παρούσα μελέτη εκπονήθηκε στα πλαίσια πτυχιακής εργασίας του Τεχνολογικού Εκπαιδευτικού Ιδρύματος Μεσολογγίου και τμήματος Τηλεπικοινωνιακών Συστημάτων και Δικτύων. Στόχος της μελέτης αυτής ήταν η πραγματοποίηση μιας Android εφαρμογής, η οποία να είναι σε θέση να λαμβάνει ήχους από το φυσικό περιβάλλον και να απεικονίζει γραφικά το φάσμα του ήχου.

Η εφαρμογή που αναπτύχθηκε ονομάστηκε Sound Visualizer. Μπορεί να τρέξει σε οποιαδήποτε Android συσκευή αρκεί το λειτουργικό να είναι τουλάχιστον έκδοσης 2.2.

Παρακάτω θα αναλυθεί η πορεία που ακολουθήθηκε για τον προγραμματισμό της εφαρμογής σε Java for Android, καθώς και βασικά στοιχεία για την ανθρώπινη ομιλία. Ο συνδυασμός αυτών των δύο γνώσεων στάθηκε θεμελιώδης για την περάτωση της παρούσας πτυχιακής εργασίας.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1 : ΛΕΙΤΟΥΡΓΙΚΟ ΣΥΣΤΗΜΑ ANDROID

Το Android είναι ένα λειτουργικό σύστημα το οποίο τρέχει τον πυρήνα του λειτουργικού Linux και αναπτύχθηκε από την εταιρεία Google για κινητές συσκευές.

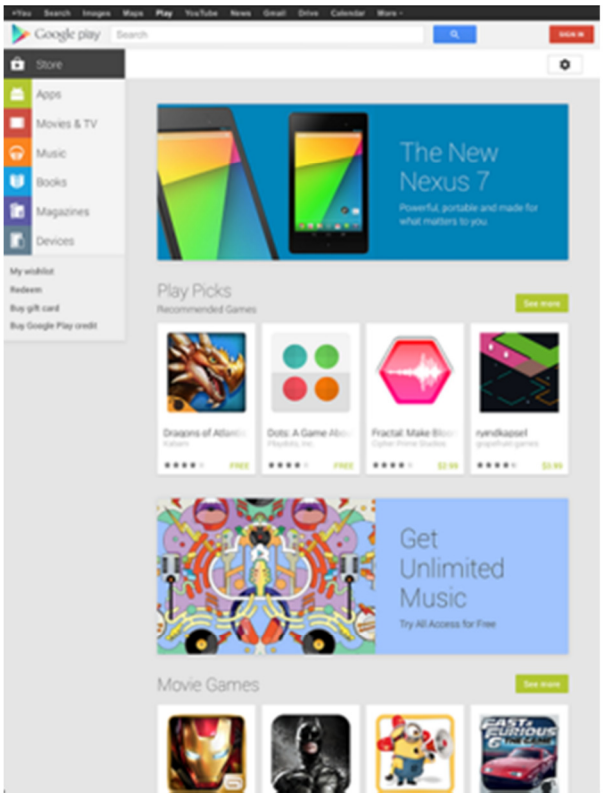
Είναι το λειτουργικό σύστημα που δίνει στους προγραμματιστές και κατασκευαστές τη δυνατότητα να μπορούν να συνθέτουν κώδικα, να επεμβαίνουν στον ήδη υπάρχον και να αναπτύσσουν έναν καινούργιο ολόκληρο κόσμο δυνατοτήτων στις κινητές συσκευές μέσω διάφορων εφαρμογών (mobile applications). Μέσω του Android, ένας κατασκευαστής μπορεί να φτιάξει από ένα ψηφιακό ρολόι που απλά θα εμφανίζει την ώρα, μέχρι μία εφαρμογή εικονικής πραγματικότητας (Virtual reality).

Βασικά χαρακτηριστικά του λειτουργικού συστήματος Android είναι ότι υποστηρίζει σύνδεση με βάσεις δεδομένων με χρήση SQLite, για ανάγκες αποθήκευσης, περιλαμβάνει ένας προσομοιωτή συσκευής, εργαλεία για διόρθωση σφαλμάτων, μνήμη και εργαλεία ανάλυσης της απόδοσης του εκτελέσιμου λογισμικού καθώς και ένα επιπρόσθετο για το Eclipse IDE και ότι υποστηρίζει Java.

Με την άφιξη του Android στο χώρο της τεχνολογίας, δημιουργήθηκε μία νέα πραγματικότητα στον κόσμο των κινητών συσκευών. Νέες θέσεις εργασίας δημιουργήθηκαν και δημιουργούνται ολοένα και περισσότερες εφαρμογές για κινητά, συνεχώς παρατηρούνται βελτιστοποιήσεις στους επεξεργαστές για να είναι πιο γρήγορη η περιήγηση στο Internet και για να μπορούν να τρέχουν πιο γρήγορα οι εφαρμογές.

Οι κινητές συσκευές που έχουν λειτουργικό σύστημα Android, έχουν τη δυνατότητα όπως αναφέρθηκε, να τρέχουν διάφορες εφαρμογές και παιχνίδια που αναφέρονται γενικά με τον όρο Android Applications (Android Apps).

Μία από τις πιο βασικές εφαρμογές που υπάρχει προεγκατεστημένη στις συσκευές με Android είναι το Google Play (παλιότερα ονομαζόταν Android Market). Παρακάτω στην εικόνα 1.1 μπορούμε να δούμε μία ενδεικτική φωτογραφία που απεικονίζει το Google Play. Η εφαρμογή αυτή είναι μια ψηφιακή πλατφόρμα διανομής εφαρμογών για το Android λειτουργικό σύστημα και ένα διαδικτυακό εργαλείο ηλεκτρονικών και ψηφιακών μέσων αποθήκευσης, που λειτουργεί από την Google. Η υπηρεσία επιτρέπει στους χρήστες να περιηγούνται και να κατεβάζουν εφαρμογές που έχουν αναπτυχθεί με το Android SDK και δημοσιεύονται μέσω του Google, καθώς και μουσική, περιοδικά, βιβλία, ταινίες και τηλεοπτικά προγράμματα. Οι χρήστες μπορούν επίσης να αγοράσουν υλικό, όπως Chromebooks, το Google Nexus-branded κινητών συσκευών, Chromecasts και αξεσουάρ, μέσω του Google Play.



Εικόνα 1.1: Google Play

Οι εφαρμογές που κατεβάζει ένας χρήστης μπορούν να αποθηκευτούν πολύ απλά, μέσω από το Google Play πατώντας το κουμπί "Εγκατάσταση". Το ίδιο απλά μπορούν και να απεγκατασταθούν από την συσκευή Android, συνήθως πιέζοντας για λίγα δευτερόλεπτα το εικονίδιο της εφαρμογής.

Δεν μπορούν όμως όλες οι εφαρμογές να απεγκατασταθούν. Για παράδειγμα το Google Play, είναι μία από τις εφαρμογές που δεν μπορεί να διαγραφεί και αυτό είναι χρήσιμο γιατί ο χρήστης μπορεί κατά λάθος να σβήσει μία εφαρμογή που δεν θα έπρεπε.

Παρακάτω θα δούμε κάποια ενδεικτικά παραδείγματα εφαρμογών, οι οποίες έχουν συναφές νόημα και στόχο λειτουργίας με το Sound Visualizer που περιγράφεται στην συγκεκριμένη πτυχιακή εργασία.

Αρχικά, ας δούμε μία πολύ ενδιαφέρουσα εφαρμογή που ονομάζεται Sound Meter. Στην εικόνα 1.2, μπορούμε να δούμε ένα στιγμιότυπο της εφαρμογής αυτής.

Εικόνα 1.2: Sound Meter

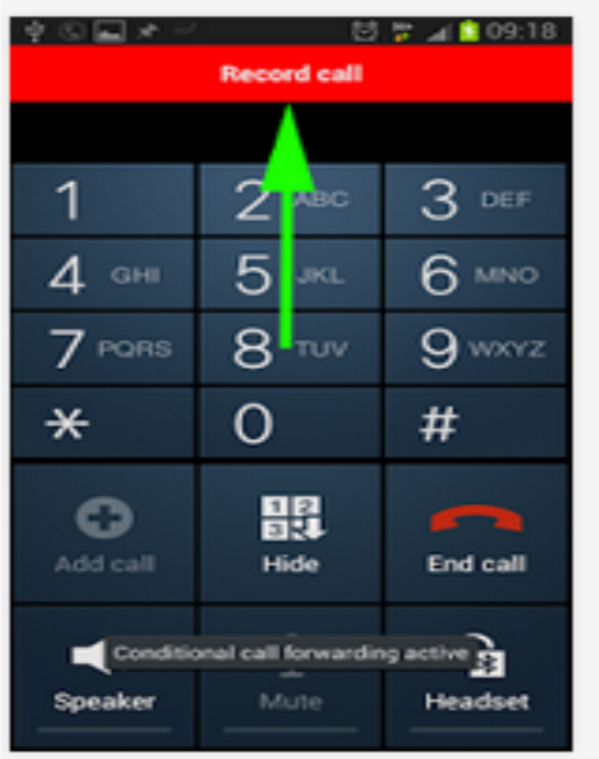
Το Sound Meter είναι ένα Android Application, το οποίο είναι σε θέση να μετράει τον ήχο του φυσικού περιβάλλοντος σε decibels. Οι προγραμματιστές που αναπτύξανε αυτήν την εφαρμογή, συνδέσανε συσκευές android με πραγματικά "ντεσιμπελόμετρα" και είδανε ότι τα αποτελέσματα ήταν ίδια με πολύ μικρή απόκλιση.

Μία πολύ σημαντική σημείωση στην προκειμένη περίπτωση είναι ότι τα μικρόφωνα είναι σχεδιασμένα για ανθρώπινη φωνή (300 – 3400 Hz, 40 – 60db). Για τον λόγο αυτό, υπάρχει περιορισμός αναγνώρισης στους πολύ υψηλούς ή πολύ χαμηλούς ήχους .

Ένα δεύτερο παράδειγμα εφαρμογής είναι το Voice Recorder, με το οποίο μπορεί ένας χρήστης να ηχογραφήσει ότι ακούει γύρω του. Στην εικόνα 1.3 μπορούμε να δούμε ένα χαρακτηριστικό στιγμιότυπο αυτής της εφαρμογής.

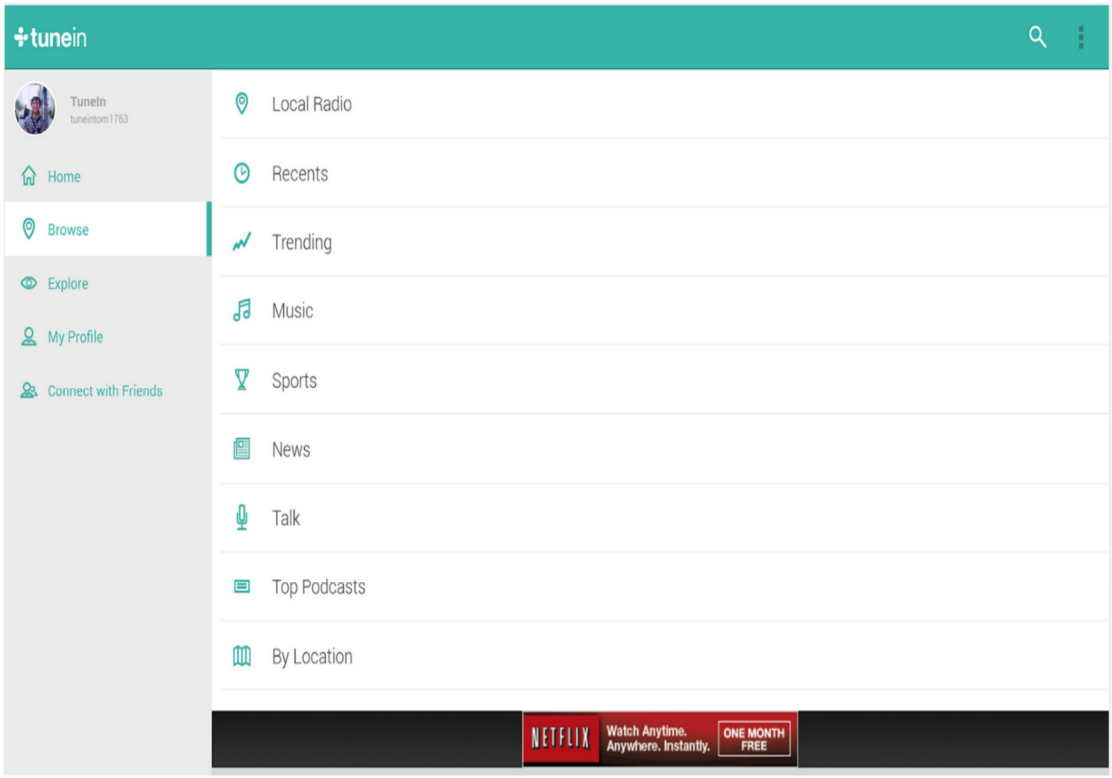
Εικόνα 1.3: Voice Recorder

Μεγάλο ενδιαφέρον έχουν δείξει οι χρήστες σε εφαρμογές που τους δίνουν τη δυνατότητα να ηχογραφούν κλήσεις. Παλαιότερα ήταν ενσωματωμένη αυτή η δυνατότητα στα περισσότερα κινητά, πλέον όμως χρειάζεται ειδική εφαρμογή για να εκτελεστεί μία τέτοια εργασία. Ένα από τα πιο γνωστά παραδείγματα τέτοιων application είναι το ACR που βλέπουμε στην εικόνα 1.4.



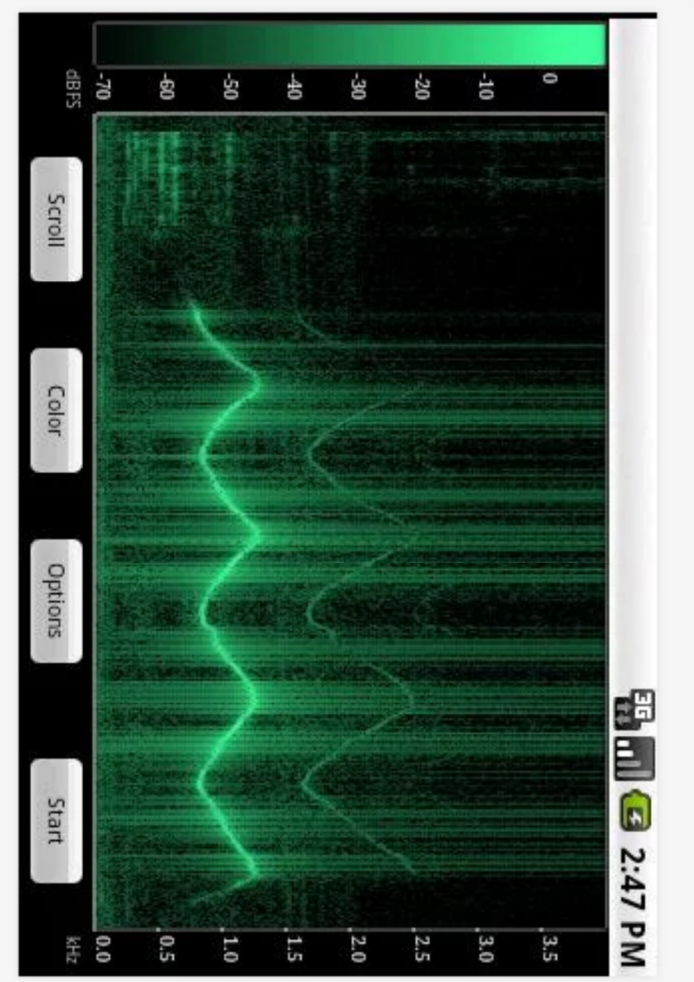
Εικόνα 1.4: ACR

Στην εικόνα 1.5, βλέπουμε μια εφαρμογή η οποία είναι σε θέση να αναζητά ραδιοφωνικούς σταθμούς και μουσική.



Εικόνα 1.5: tunein android

Τέλος, στην εικόνα 1.6 μπορούμε να δούμε μία εφαρμογή που θυμίζει την εφαρμογή που αναπτύχθηκε στα πλαίσια της παρούσας πτυχιακής εργασίας, η οποία ονομάζεται Spectral Audio Analyzer και απεικονίζει γραφικά το φάσμα του ήχου. Χαρακτηριστική της λειτουργίας του είναι η εικόνα 1.6.



Εικόνα 1.6: Spectral Audio Analyzer

Όπως βλέπουμε παραπάνω, υπάρχουν εφαρμογές οι οποίες μπορούν να επεξεργαστούν τον ήχο. Τα σήματα φωνής δηλαδή που υπάρχουν στο φυσικό περιβάλλον. Η ανάλυση του ήχου και η επεξεργασία του αρχικά ίσως φαίνεται μια απλοϊκή υπόθεση, και όχι τόσο σημαντική, στην πραγματικότητα όμως μπορεί να έχει πάρα πολλές εφαρμογές σε διάφορα τμήματα της τεχνολογίας.

Ένα καλά σχεδιασμένο σύστημα ανάλυσης ήχου, για παράδειγμα μπορεί να χρησιμοποιηθεί σε συστήματα ελέγχου ασφαλείας, μπορεί να χρησιμοποιηθεί σε εφαρμογές που δίνουν τη δυνατότητα στο χρήστη να δίνει φωνητικές εντολές στην κινητή συσκευή και αυτή να καταλαβαίνει και να "ακούει" τον χρήστη. Ακόμα, υπάρχουν εφαρμογές που μπορεί να ηχογραφούν το εύρος του ροχαλητού του ανθρώπου όταν αυτός κοιμάται και να εξάγει ακόμα και ιατρικά συμπεράσματα!

Το Android λοιπόν, δίνει τη δυνατότητα σύλληψης και επεξεργασίας του ήχου και των σημάτων φωνής. Σύμφωνα με ορισμό του wikipedia, μπορούμε να περιγράψουμε τον ήχο ως εξής: "Ο ήχος είναι η αίσθηση του προκαλείται λόγω της διέγερσης των αισθητηρίων οργάνων της ακοής από μεταβολές πίεσης του ατμοσφαιρικού αέρα. Αυτές οι μεταβολές διαδίδονται με τη μορφή ηχητικών κυμάτων . Πολλές φορές στην πράξη, ο όρος χρησιμοποιείται ως ταυτόσημος με την έννοια των ηχητικών κυμάτων - για παράδειγμα, συνηθίζεται η έκφραση διάδοση του ήχου(αντί του ορθότερου *διάδοση των ηχητικών κυμάτων*). "

Όπως σε κάθε λειτουργικό σύστημα, έτσι και στο Android, κάποιοι προγραμματιστές έχουν δουλέψει για να δημιουργήσουν έργα με βασικές λειτουργίες, όπως το άνοιγμα της κάμερας του κινητού, το άνοιγμα του μικροφώνου μέσα από κάποιο πρόγραμμα και διάφορα άλλα.

Έτσι και στη δική μας περίπτωση, υπάρχουν έτοιμα έργα προγραμματιστών, που ονομάζονται βιβλιοθήκες ή έτοιμες κλάσεις και δίνουν τη δυνατότητα στους προγραμματιστές να μπορούν να τις χρησιμοποιούν για να συλλάβουν ήχο με τα μέσα που διαθέτει η συσκευή Android (μικρόφωνο).

Υπάρχουν δύο έτοιμες κλάσεις που δίνουν τη δυνατότητα σύλληψης ήχου στο Android. Η AudioRecord και η MediaRecorder.

Θα αναλύσουμε παρακάτω και τις δύο κλάσεις και θα δούμε τις διαφορές τους.

Η κλάση AudioRecord διαχειρίζεται τους πόρους του ήχου σε εφαρμογές Java για την καταγραφή ήχου από το υλικό εισόδου της πλατφόρμας. Αυτό επιτυγχάνεται μέσω της ανάγνωσης των δεδομένων από την κλάση AudioRecord. Η επιλογή της μεθόδου που θα χρησιμοποιηθεί θα βασίζεται στην μορφή ήχου αποθήκευσης δεδομένων που είναι η πιο βολική για τον χρήστη της AudioRecord.

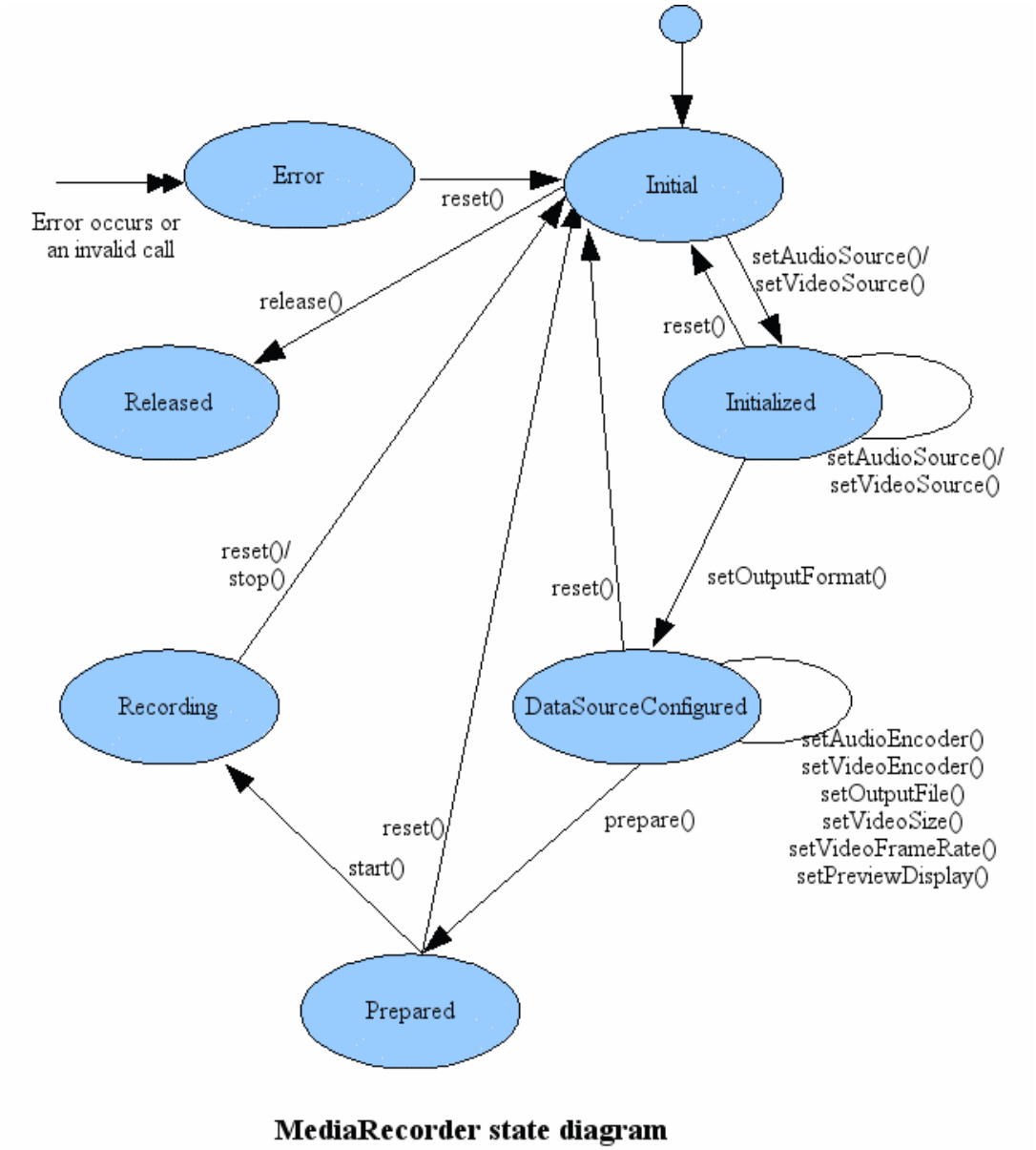
Μετά τη δημιουργία του, ένα αντικείμενο AudioRecord προετοιμάζει buffer ήχου, που θα γεμίσει με τα νέα δεδομένα ήχου. Το μέγεθος του buffer, που ορίζεται κατά τη διάρκεια της κατασκευής, καθορίζει πόσο καιρό ένα αντικείμενο AudioRecord μπορεί να καταγράψει πριν από τα δεδομένα που δεν έχει διαβάσει ακόμα. Τα δεδομένα θα πρέπει να διαβαστούν από το υλικό ήχου σε κομμάτια των μεγεθών κατώτερο από το συνολικό μέγεθος του buffer εγγραφής.

Η έτοιμη αυτή κλάση έχει φυσικά και έτοιμες μεθόδους, όπως η startRecording(), η οποία σηματοδοτεί την έναρξη της ηχογράφησης, η stopRecording(), η οποία σταματάει την ηχογράφηση και διάφορες άλλες που θα δούμε αναλυτικά αργότερα.

Η δεύτερη έτοιμη κλάση που υπάρχει για την σύλληψη ήχου είναι η MediaRecorder. Η κλάση αυτή έχει παρόμοιες έτοιμες μεθόδους με την AudioRecord, αλλά γενικά

χρησιμοποιείται τόσο για χειρισμό του μικροφώνου της συσκευής όσο και για το χειρισμό της κάμερας. Για το λόγο αυτό, προτιμήθηκε στην παρούσα εργασία η AudioRecord.

Στην εικόνα 1.7 μπορούμε να δούμε ένα διάγραμμα ροής για την MediaRecorder.



Εικόνα 1.7: Media Recorder, διάγραμμα ροής

Όπως παρατηρούμε, από την αρχική κατάσταση (Initial) η κλάση MediaRecorder, μεταβαίνει στην κατάσταση 2 (Initialized) μόλις κληθεί η συνάρτηση setAudioSource(). Αυτή ρυθμίζει την πηγή ήχου που πρόκειται να χρησιμοποιηθεί για την καταγραφή. Το ίδιο φυσικά συμβαίνει με την setVideoSource(), που ρυθμίζει αντίστοιχα την πηγή video.

Για να ξεκινήσει η κατάσταση 3 (DataSourceConfigured) καλούνται οι μέθοδοι setAudioEncode() ή setVideoEncoder(), οι οποίες ορίζουν αντίστοιχα τον κωδικοποιητή ήχου ή βίντεο που πρόκειται να χρησιμοποιηθεί για την καταγραφή.

Η setOutptuFile(), η οποία καθορίζει την περιγραφή του αρχείου που θα γραφτεί, η setVideoSize() και setVideoFrameRate(), οι οποίες καλούνται μόνο στην περίπτωση που θέλουμε να καταγράψουμε βίντεο και η πρώτη καθορίζει το μέγεθος του βίντεο και η δεύτερη το εύρος του καρέ του βίντεο. Τέλος καλείται η setPreviewDisplay(),η οποία ρυθμίζει την επιφάνεια που θα εμφανιστεί αυτό που καταγράφεται (πάλι στην περίπτωση του βίντεο).

Αφού κληθούν όλες αυτές οι συναρτήσεις η MediaRecorder είναι έτοιμη να ξεκινήσει την ηχογράφηση ή τη σύλληψη βίντεο.

Παρόμοια λειτουργεί και η AudioRecord.

ΚΕΦΑΛΑΙΟ 2 : ΗΧΗΤΙΚΑ ΚΥΜΑΤΑ ΚΑΙ ΙΔΙΑΙΤΕΡΟΤΗΤΕΣ ΤΩΝ ΣΗΜΑΤΩΝ ΦΩΝΗΣ

Τα ηχητικά κύματα είναι διαμήκη κύματα (πυκνώματα - αραιώματα της πυκνότητας και της πίεσης του αέρα) και ανάλογα με τη συχνότητά τους διακρίνονται σε ήχους, υπερήχους και υποήχους.

Οι ήχοι είναι ηχητικά κύματα συχνότητας μεταξύ 20 και 20000 Hz. Οι ήχοι γίνονται αντιληπτοί από το αισθητήριο ακοής του ανθρώπου, αφού γίνει η κατάλληλη βιοφυσική μετατροπή των μηχανικών ταλαντώσεων σε ηλεκτρικά σήματα, τα οποία επεξεργάζεται έπειτα ο ανθρώπινος εγκέφαλος.

Οι υπέρηχοι είναι ηχητικά κύματα συχνότητας f μεγαλύτερης των 20000 Hz, η οποία είναι το άνω όριο των ακουστικών συχνοτήτων. Συχνότητες της περιοχής συχνοτήτων των υπερήχων γίνονται αντιληπτές από άλλους ζωντανούς οργανισμούς, π.χ. τους σκύλους. Οι υπέρηχοι έχουν μεγάλη κατευθυντικότητα και χρησιμοποιούνται σε μια σειρά διαγνωστικών και άλλων τεχνολογικών εφαρμογών, όπως π.χ. οι βυθομετρήσεις, ο ηχοεντοπισμός, η ιατρική διάγνωση και θεραπεία, η ακουστική διάγνωση ρωγμών ή φαινομένων σπηλαίωσης σε μέταλλα, ο κατακερματισμός ανεπιθύμητων επικαλύψεων σε ορυκτά (φυσικά ή καλλιτεχνικά δημιουργήματα) και ο καθαρισμός εργαλείων (υπερηχητικά “πλυντήρια”).

Οι υπόηχοι είναι ηχητικά κύματα συχνότητας f μικρότερης των 20 Hz, η οποία είναι το κάτω όριο των ακουστικών συχνοτήτων. Οι υπόηχοι προκαλούνται από σεισμούς, ηφαίστεια, βροντές, ανέμους και βαριές μηχανολογικές εγκαταστάσεις. Συχνότητες της περιοχής συχνοτήτων των υποήχων γίνονται αντιληπτές από άλλους ζωντανούς οργανισμούς, π.χ. τις κατσαρίδες (βιολογική μέθοδος ανίχνευσης σεισμών!).

Οι φυσιολογικοί ήχοι της ομιλίας παράγονται με τη ρύθμιση της ροής του εκπνεόμενου αέρα. Για την παραγωγή των περισσοτέρων από τους ήχους οι πνεύμονες παρέχουν ρεύμα αέρα, το οποίο περνά μέσω των φωνητικών χορδών (πτυχών), που αναφέρονται και ως γλωττίδα, προκαλώντας την ταλάντωσή τους, με αποτέλεσμα τη ρύθμιση της ροής του αέρα. Ο αέρας στη συνέχεια διέρχεται από κοιλότητες των φωνητικών οργάνων, πριν βγει από το σώμα, από το στόμα και σε μικρό βαθμό από τη μύτη. Οι ήχοι που παράγονται με αυτόν τον τρόπο είναι τα φωνήεντα (φωνήεις ήχοι), ενώ οι ήχοι που παράγονται στη στοματική κοιλότητα, χωρίς τη χρήση των φωνητικών χορδών, ονομάζονται άφωνοι ήχοι και αντιστοιχούν στα σύμφωνα.

Η ανθρώπινη ομιλία είναι πολύ βασικό κομμάτι φυσικά και στις εφαρμογές κινητών συσκευών. Αυτό γιατί είναι απαραίτητη για να υπάρχει νόημα στη λειτουργία των κινητών συσκευών η ψηφιοποίηση της ανθρώπινης ομιλίας. Σύμφωνα με έρευνες η μετατροπή ενός σήματος φωνής σε ψηφιακό απαιτεί δειγματοληψία με συχνότητα τουλάχιστον 8 kHz και χρησιμοποίηση πάνω από 10 bits ανά δείγμα, δεδομένου πως το συχνοτικό περιεχόμενο των σημάτων της ανθρώπινης ομιλίας μπορεί να θεωρηθεί πως δεν έχει ενέργεια σε συχνότητες πάνω από τα 4 kHz. Για τη μετάδοση ενός δευτερολέπτου ομιλίας απαιτούνται έτσι 8000 δείγματα ή περίπου 10 Kbytes δεδομένων.

***Εύφωνοι ήχοι*:** Κατά την παραγωγή των ήχων αυτών οι φωνητικές

χορδές πάλλονται περιοδικά και αφήνουν να περάσει από μέσα τους ένα

περιοδικό κύμα αέρα. Ο αέρας αυτός στη συνέχεια περνάει από τη φαρυγγική και τη στοματική κοιλότητα η μορφολογία των οποίων διαμορφώνουν τον τελικό ήχο. Η περίοδος ταλάντωσης των φωνητικών χορδών ονομάζεται θεμελιώδης συχνότητα (pitch). Παραδείγματα εύφωνων ήχων αποτελούν τα φωνήεντα.

***Άφωνοι ήχοι*:** Κατά την παραγωγή άφωνων ήχων οι φωνητικές χορδές παραμένουν διαρκώς ανοικτές επιτρέποντας να περάσει από μέσα τους ένα συνεχές κύμα αέρα. Στη συνέχεια, ο αέρας περνάει από τη φαρυγγική και τη στοματική κοιλότητα όπου έχουν δημιουργηθεί διάφορες στενώσεις της ακουστικής οδού. Η θέση των στενώσεων αυτών καθορίζουν τον τελικό ήχο. Παραδείγματα άφωνων ήχων αποτελούν τα σύμφωνα σίγμα, θήτα, φι και άλλα.

ΚΕΦΑΛΑΙΟ 3 : Η ΘΕΩΡΙΑ ΓΙΑ ΤΗΝ ΥΛΟΠΟΙΗΣΗ ΚΑΙ Η ΥΛΟΠΟΙΗΣΗ

Για την υλοποίηση οποιασδήποτε εφαρμογής ή για τη λύση οποιουδήποτε προβλήματος γενικότερα, το πιο σημαντικό κομμάτι είναι η έρευνα για τα εργαλεία που θα χρησιμοποιηθούν.

Έτσι και στην περίπτωση της εφαρμογής που αναπτύχθηκε για την γραφική απεικόνιση του φάσματος του ήχου, το πρώτο πράγμα που έπαιξε ρόλο στην ανάπτυξή της ήταν η επιλογή των εργαλείων και της γλώσσας ανάπτυξης.

Ως αποτέλεσμα της έρευνας, αποφασίστηκε ότι η γλώσσα ανάπτυξης της εφαρμογής θα είναι η Java for Android, που είναι ένας συνδυασμός της γλώσσας Java με xml και το περιβάλλον ανάπτυξης το Eclipse.

Το Eclipse είναι ένα περιβάλλον ανάπτυξης, το οποίο υποστηρίζει πάρα πολλές γλώσσες όπως Java, C, C++ και άλλες.

Για να υποστηρίξει όμως Java for Android χρειάζεται ένα ειδικό plugin το ADT Plugin (Android Development Tool), το οποίο μπορεί κανείς να το κατεβάσει από την επίσημη ιστοσελίδα του Android developers (http://developer.android.com/tools/sdk/eclipse-adt.html).

Έτσι λοιπόν, το ADT Plugin επεκτείνει τις δυνατότητες του Eclipse για να επιτρέπει στους προγραμματιστές να αναπτύσσουν γρήγορα και εύκολα Android projects, να δημιουργούν εφαρμογές με διεπαφές χρήστη (user interfaces - UI), να προσθέσουν πακέτα με βάση το API του Android frame, να διορθώνουν τις εφαρμογές τους, χρησιμοποιώντας τα

εργαλεία του Android SDK , ακόμα και να εξάγουν .Apk αρχεία προκειμένου να διανείμουν τις εφαρμογές τους.

Το plugin που αναλύεται παραπάνω, δίνει τη δυνατότητα στο Eclipse να αναπτύσσει εικονικές συσκευές Android (Android Virtual Devices AVDs). Οι εικονικές συσκευές είναι απαραίτητες για να μπορούν οι προγραμματιστές να τρέχουν τις εφαρμογές τους.

Στην εικόνα 3.1 μπορούμε να δούμε πώς είναι η εικονική μηχανή Android.

Εικόνα 3.1 : Android Virtual Device

Εκτός φυσικά από τις εικονικές συσκευές Android, το ADT Plugin σε συνδυασμό με άλλα εργαλεία που θα αναλυθούν στη συνέχεια δίνει τη δυνατότητα αντί να δημιουργεί κάποιος μία εικονική συσκευή Android, να συνδέσει τη δική του συσκευή Android με το Eclipse και να τρέχει εκεί τις εφαρμογές του.

Πιο αναλυτικά, τα πρώτα βήματα που ακολουθήθηκαν για την ανάπτυξη της εφαρμογής ήταν τα εξής:

ΒΗΜΑΤΑ ΓΙΑ ΤΗΝ ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ANDROID

ΕΓΚΑΤΑΣΤΑΣΗ ΠΕΡΙΒΑΛΛΟΝΤΟΣ

1) Κατεβάζουμε το Eclipse από το παρακάτω link http://www.eclipse.org/downloads/

2) Κάνουμε εξαγωγή το αρχείο σε φάκελο που ονομάζουμε Eclipse

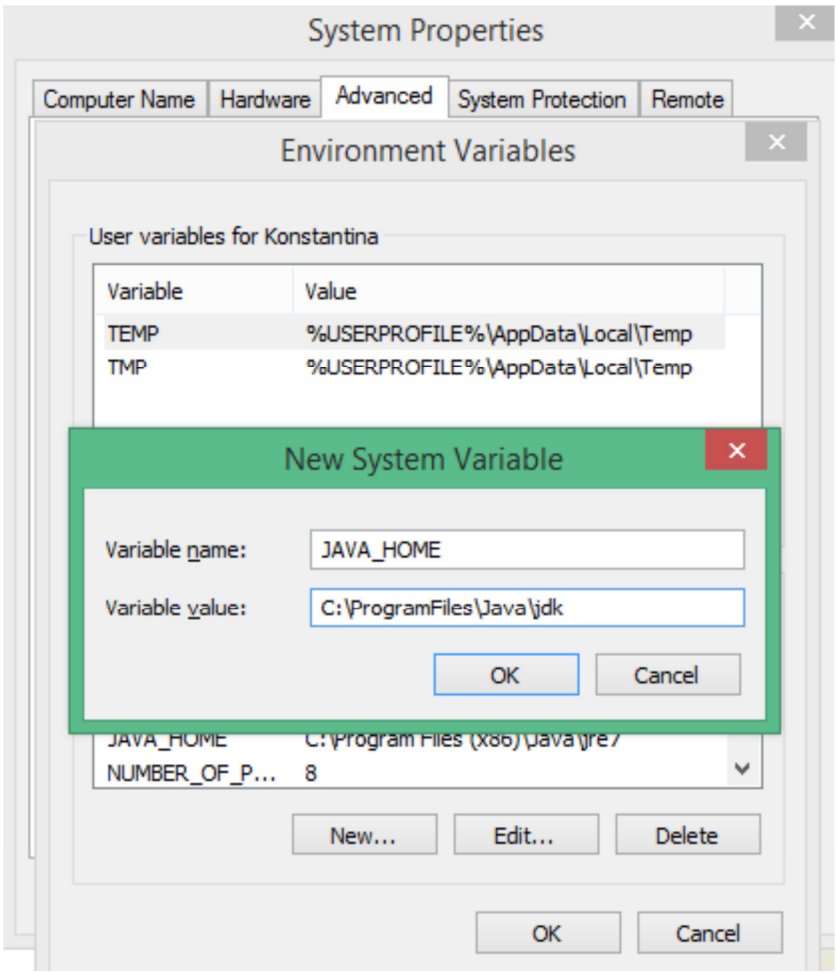
3) Μόλις αποσυμπιεστεί, ανοίγουμε το φάκελο και πηγαίνουμε εκεί που υπάρχει το αρχείο eclipse.exe.

4) Εάν ανοίξουμε αυτή τη στιγμή το eclipse θα πάρουμε μήνυμα σφάλματος πως δεν υπάρχει JRE και ότι δεν βρέθηκε το javaw στο PATH.

Για να λυθεί αυτό

5) Κατεβάζουμε το JRE (Java Runtime Environment) από http://www.oracle.com/technetwork/java/javase/downloads/index.html

6) Κάνοντας δεξί κλικ στο κατάλογο C:, επιλέγουμε Properties (Ιδιότητες) -> Advanced System Settings → Environment Variables και κάτω στο κουμπί New φτιάχνουμε μία νέα μεταβλητή με όνομα JAVA_HOME και Path τη διαδρομή στο σκληρό μας που βρίσκεται το jdk (jre). Η εικόνα 3.2 απεικονίζει πώς φτιάχνουμε την μεταβλητή JAVA_HOME



Εικόνα 3.2: Δημιουργία Μεταβλητής Περιβάλλοντος JAVA_HOME

7) Με παρόμοιο τρόπο βρίσκουμε τη μεταβλητή Path, την επιλέγουμε, πατάμε edit.. και προσθέτουμε ένα ερωτηματικό (;) και στη συνέχεια το φάκελο bin του jdk, για παράδειγμα C:\ProgramFiles\Java\jdk\bin

8) Αποθηκεύουμε τις αλλαγές μας.

9) Το Eclipse μπορεί πλέον να λειτουργήσει.

ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗ ΤΟΥ ECLIPSE ΜΕ ΤΟ ANDROID SDK

1. Ανοίγουμε το Eclipse.
2. Επιλέγουμε Help → Install New Software.
3. Πατάμε Add επάνω δεξιά
4. Στο πλαίσιο που εμφανίζεται βάζουμε “ADT Plugin” για όνομα (name)
5. Στο Url βάζουμε https://dl-ssl.google.com/android/eclipse/
6. Κλικ OK.
7. Στο πλαίσιο των Διαθέσιμων Software, επιλέγουμε Developer Tools
8. Πατάμε Next
9. Accept the license agreement και Next
10. Όταν ολοκληρωθεί η εγκατάσταση κάνουμε επανεκκίνηση το Eclipse
11. Μόλις ανοίξει το Eclipse θα υπάρχει ένα μήνυμα για εγκατάσταση του Android SDK Manager.
12. Επιλέγουμε Yes στα μηνύματα και κάνουμε εγκατάσταση το Android SDK Manager.

ΣΥΝΔΕΣΗ ANDROID ΣΥΣΚΕΥΗΣ ΣΤΟ ECLIPSE

1. Πρέπει να ενεργοποιήσουμε το USB debugging
2. Πάμε στις Ρυθμίσεις του κινητού -> Εφαρμογές -> Ανάπτυξη
3. Τσεκ στο “USB Debugging” (Εντοπισμός Σφαλμάτων)
4. Πρέπει να κατεβάσουμε το usb driver της συσκευής.
5. Αυτό το επιλέγουμε από http://developer.android.com/tools/extras/oem-usb.html με βάση τη συσκευή μας.
6. Αφού εγκαταστήσουμε και το usb driver πλέον, μπορούμε να πάμε στο Eclipse, να κάνουμε ένα Android Project (File → New → Project → Android → Android Application Project
7. Βάζουμε το όνομα που θέλουμε (πχ Sound Analyzer) και επιλέγουμε την κατάλληλη έκδοση Android που έχει η συσκευή μας (πχ. 2,1)
8. Next
9. Next
10. Επιλέγουμε το εικονίδιο που θέλουμε να έχει η εφαρμογή μας με το κουμπί Browse..
11. Next
12. Επιλέγουμε Blank Activity
13. Finish
14. Μέχρι αυτό το σημείο δημιουργήσαμε μία απλή εφαρμογή.

15. Πάμε δεξί κλικ πάνω στο Project, Run as → Android Application

16. Θα πρέπει να έχει βγάλει ως device την συσκευή μας

17. Οκ

18. Η εφαρμογή τρέχει στο κινητό και εμφανίζει “Hello World”!

Μετά από τα παραπάνω, έχουμε φτάσει στο σημείο που μπορούμε να τρέξουμε ένα πολύ απλό πρόγραμμα Android. Και όπως έχει καθιερωθεί στον κόσμο των προγραμματιστών, το πρώτο πρόγραμμα σε κάθε γλώσσα είναι το Hello Word!.

Στην συνέχεια με βάση τις οδηγίες που υπάρχουν στο επίσημο site των android developers, αναλύουμε πώς φτιάχνουμε ένα Android Project στο Eclipse.

Αρχικά πατάμε New πάνω δεξιά στο Eclipse και επιλέγουμε το Android Project. Είναι χρήσιμο στο σημείο αυτό να αναφέρουμε τι πρέπει να συμπληρωθεί για να δημιουργηθεί ένα Android Project και τι σημαίνει κάθε τιμή που μας ζητάει το Eclipse.

- **Application name:** Το όνομα της εφαρμογής
- **Project name:** Το όνομα του καταλόγου του Project
- **Package name:** Το όνομα του πακέτου που θα περιλαμβάνει τις κλάσεις
- **Minimum Required SDK:** Είναι η χαμηλότερη έκδοση του Android που μπορεί να υποστηρίξει η εφαρμογή
- **Target SDK:** Είναι η υψηλότερη έκδοση του Android που μπορεί να υποστηρίξει η εφαρμογή που θα αναπτυχθεί
- **Compile With:** Είναι η πλατφόρμα με την οποία θα γίνει compile την εφαρμογή

- **Theme:** Είναι το στυλ του Android user interface που θα ακολουθήσει η εφαρμογή.

Εάν συμπληρώσουμε αυτά, στα επόμενα βήματα το Eclipse, παροτρύνει το χρήστη να δώσει όνομα για την κύρια κλάση του Project , την κύρια διεργασία δηλαδή (MainActivity) και να επιλέξει εικονίδιο με το οποίο θα εμφανίζεται σαν συντόμευση στην κινητή Android συσκευή.

Σημαντικό επίσης είναι στο σημείο αυτό, να δούμε τι περιέχει ένα android project μόλις φτιάχνεται.

Έτσι λοιπόν περιέχει:

- Το *AndroidManifest.xml*, το οποίο είναι το μανιφέστο της εφαρμογής, περιγράφει δηλαδή τις βασικές αρχές της εφαρμογής, όπως το minimum required sdk, το target sdk και άλλα.
- Ένα φάκελο src/, ο οποίος θα περιέχει τα source files (αρχεία κώδικα)
- Ένα φάκελο res/ , ο οποίος θα περιέχει τους πόρους της εφαρμογής, για παράδειγμα το εικονίδιο της εφαρμογής, διάφορες φωτογραφίες και άλλα.

Για την ανάπτυξη της εφαρμογής, δημιουργήθηκαν τέσσερις κλάσεις οι: MainActivity, που είναι η κύρια δραστηριότητα της εφαρμογής, η RecordSound, η οποία είναι η κλάση που ηχογραφεί τον ήχο, η DrawSound, που απεικονίζει γραφικά το φάσμα του ήχου και η Sleeper, η οποία είναι η κλάση που χειρίζεται τα νήματα της εφαρμογής.

Πιο αναλυτικά, μπορούμε να ξεκινήσουμε από την Sleeper. Η κλάση αυτή είπαμε ότι χειρίζεται τα νήματα της εφαρμογής.

Αρχικά, καλό είναι να παραθέσουμε τον ορισμό του νήματος:

"Στην πληροφορική, ένα νήμα εκτέλεσης είναι η μικρότερη ακολουθία προγραμματισμένων εντολών που μπορεί να διαχειρισθεί ανεξάρτητα, από το λειτουργικό σύστημα. Ένα νήμα είναι μια ελαφριά διεργασία. Η υλοποίηση των νημάτων και των διεργασιών διαφέρει από το ένα λειτουργικό σύστημα στο άλλο. Στις περισσότερες όμως περιπτώσεις ένα νήμα εμπεριέχεται σε μια διεργασία. Μπορούν να υπάρχουν πολλαπλά νήματα μέσα στην ίδια διεργασία τα οποία μπορούν να μοιράζονται πόρους από το σύστημα, όπως μνήμη. Διαφορετικές διεργασίες δεν μπορούν να μοιράζονται τους ίδιους πόρους. Συγκεκριμένα, τα νήματα μιας διεργασίας περιέχουν τις εντολές προς την εκτελούμενη διεργασία (δηλαδή τον κώδικα της) και το εννοιολογικό της πλαίσιο (οι τιμές των μεταβλητών της σε οποιαδήποτε χρονική στιγμή)."

Έτσι λοιπόν αυτό που κάνει η κλάση Sleeper είναι αυτή που κάνει το νήμα της εφαρμογής να περιμένει όσο η κλάση RecordSound συλλέγει δεδομένα του ήχου. Αυτό πρακτικά χρησιμεύει στο εξής: Έχουμε την κλάση RecordSound και DrawSound. Όπως είπαμε, η RecordSound ηχογραφεί και η DrawSound απεικονίζει γραφικά το φάσμα του ήχου. Πρέπει λοιπόν να υπάρχει μία διαδικασία που να εξασφαλίζει ότι έχει συλληφθεί ο ήχος προτού ξεκινήσει η απεικόνιση του. Την εργασία αυτή έρχεται να επιτελέσει ένα νήμα το οποίο "κοιμάται" όσο η RecordSound συλλέγει ήχο, και "ξυπνάει" μόλις η RecordSound έχει έτοιμα τα αποτελέσματα συλλογής.

Η κλάση Sleeper εφαρμόζει (implements) ένα interface, που ονομάζεται Runnable. Το interface αυτό αντιπροσωπεύει μια εντολή που μπορεί να εκτελεστεί. Συχνά χρησιμοποιείται για να τρέξει κώδικα σε ένα διαφορετικό νήμα.

Η κλάση Sleeper αποτελείται από τα εξής χαρακτηριστικά – ιδιότητες:

- Μία μεταβλητή απόφασης (boolean), που ονομάζεται done
- Μία μεταβλητή κλάσης της MainActivity που ονομάζεται mainActivity
- Μία μεταβλητή κλάσης της RecordSound που ονομάζεται recordSound

Η κλάση Sleeper έχει έναν κατασκευαστή με ορίσματα ένα στιγμιότυπο της κλάσης MainActivity, και ένα της RecordSound.

Τέλος, η κλάση Sleeper έχει μία μέθοδο που ονομάζεται run() και η οποία καλεί την κλάση RecordSound για να ξεκινήσει να ηχογραφεί και μέχρι η RecordSound να στείλει σήμα ότι τελείωσε με την ηχογράφηση θέτει το νήμα σε sleep (να "κοιμάται").

Η κλάση RecordSound, είναι η κλάση που ηχογραφεί. Αποτελείται από τα εξής χαρακτηριστικά – ιδιότητες:

- int SAMPPERSEC = 44100
- static short[] buffer, ένας πίνακας από short τιμές για το buffer του ήχου
- AudioRecord ar, στιγμιότυπο της έτοιμης κλάσης AudioRecord
- int audioEncoding = 2
- int buffersizebytes, το μέγεθος του buffer
- int buflen
- int channelConfiguration = 16
- int mSamplesRead

- Boolean m_bDead = Boolean.valueOf(false);
- Boolean m_bDead2 = Boolean.valueOf(true);
- Boolean m_bRun
- Boolean m_bSleep = Boolean.valueOf(false);
- MainActivity m_ma, στιγμιότυπο της MainActivity
- Thread recordingThread, το νήμα ηχογράφησης

Η RecordSound αποτελείται από έναν κατασκευαστή που δέχεται ως όρισμα ενα στιγμιότυπο της MainActivity.

Μία από τις βασικότερες μεθόδους της είναι η **Init()**, η οποία φτιάχνει ένα στιγμιότυπο της έτοιμης κλάσης AudioRecord, ως εξής:

```
ar = new AudioRecord(1, 44100, channelConfiguration, audioEncoding,
AudioRecord.getMinBufferSize(SAMPPERSEC, channelConfiguration, audioEncoding));
```

Το στιγμιότυπο της κλάσης αυτής φτιάχνεται έτσι όπως αναφέρθηκε παραπάνω γιατί η έτοιμη κλάση AudioRecord, έχει έναν κατασκευαστή

```
public AudioRecord(int audioSource, int sampleRateInHz, int channelConfig, int audioFormat, int
bufferSizeInBytes)
```

Παράμετροι του κατασκευαστή:

- *audioSource*: Πηγή του ήχου. (audioSource = 1, σημαίνει ότι χρησιμοποιούμε το μικρόφωνο της συσκευής)

- *sampleRateInHz*: το ποσοστό του δείγματος εκφρασμένο σε Hertz. Τα 44100Hz είναι σήμερα το μοναδικό ποσοστό που είναι εγγυημένο ότι μπορεί να δουλεύει σε όλες τις συσκευές.
- *ChannelConfig*: περιγράφει τη διαμόρφωση των καναλιών ήχου
- *audioFormat:* Η μορφή του ήχου
- *bufferSizeInBytes:* το συνολικό μέγεθος (σε bytes) του buffer όπου τα δεδομένα ήχου είναι γραμμένα κατά τη διάρκεια της εγγραφής

Η μέθοδος **Restart(),** είναι η μέθοδος που με βάση τον έλεγχο κάποιων μεταβλητών απόφασης (boolean), έχει τη δυνατότητα να καλεί ξανά και ξανά την Init() έτσι ώστε να δημιουργεί το στιγμιότυπο της κλάσης AudioRecord, αφού πρώτα αποδεσμεύσει το προηγούμενο στιγμιότυπο. Στη συνέχεια, εάν φτιαχτεί το στιγμιότυπο όντως, τότε θα κληθεί η μέθοδος startRecording() και η startSampling(), που θα αναλυθεί στη συνέχεια η λειτουργία τους. Πρακτικά, η Restart(), είναι μία μέθοδος απαραίτητη για την εφαρμογή γιατί ηχογραφεί συνεχώς από τη στιγμή που ξεκινάει και μετά.

Η μέθοδος **StartRecording(),** εάν το στιγμιότυπο της κλάσης AudioRecord είναι κενό, ξανακαλεί την Init() για να το φτιάξει, αλλιώς καλεί την μέθοδο της έτοιμης κλάσης AudioRecord, startRecording() η οποία θα ξεκινήσει την ηχογράφηση.

Η μέθοδος **StartSampling(),** είναι η μέθοδος που με βάση κάποιες μεταβλητές απόφασης χειρίζεται το νήμα της εφαρμογής, βλέπει δηλαδή εάν πρέπει να "κοιμηθεί" ή να "ξυπνήσει".

Τέλος, η **StopRecording()**, είναι η μέθοδος που σταματάει την ηχογράφηση.

Η κλάση **DrawSound** είναι αυτή που σχεδιάζει το φάσμα του ήχου. Οτιδήποτε έχει γραφικό αποτέλεσμα στην οθόνη πρέπει να κληρονομεί την κλάση SurfaceView, η οποία παρέχει την ειδική επιφάνεια σχεδίασης. Επίσης, εφαρμόζει το interface SurfaceHolder.Callback, το οποίο λαμβάνει πληροφορίες σχετικά με την επιφάνεια σχεδίασης.

Η κλάση αυτή αποτελείται από τα εξής χαρακτηριστικά – ιδιότητες:

- Context mContext
- CDrawThread mDrawThread, κλάση CdrawThread που δηλώνεται μέσα στην DrawSound
- SurfaceHolder mHolder

Οι μέθοδοι SurcfaceChanged(), SurfaceCreated() και SurfaceDestroyed() είναι μέθοδοι που πρέπει να υλοποιηθούν γιατί η DrawSound εφαρμόζει το interface SurfaceHolder.Callback.

Η **SurfaceChanged**(), καλείται όταν αλλάζει κάτι στην επιφάνεια σχεδίασης και θέτει το μέγεθος της επιφάνειας.

Η **SurfaceCreated**(), είναι η μέθοδος που κατασκευάζει την επιφάνεια σχεδίασης και θέτει true σε μια μεταβλητή απόφασης έτσι ώστε να γνωρίζει το υπόλοιπο πρόγραμμα ότι η επιφάνεια σχεδίασης δημιουργήθηκε. Αφού γίνει true αυτή η μεταβλητή θα κληθεί η μέθοδος Restart().

Τέλος, η **SurfaceDestroyed**(), είναι η μέθοδος που καταστρέφει την επιφάνεια σχεδίασης.

Η μέθοδος **Restart**(), είναι η μέθοδος που καλείται κάθε φορά που συλλέγονται δεδομένα ήχου και μπορούν να σχεδιαστούν στην επιφάνεια σχεδίασης. Στην ουσία εφόσον

οι συνθήκες το επιτρέπουν, η **Restart**() θα καλεί την κλάση **CdrawThread** που περιγράφεται παρακάτω.

Η **CdrawThread** είναι η κλάση που σχεδιάζει. Ο κατασκευαστής αυτής της κλάσης, λειτουργεί ως εξής' παραμετροποιεί την μεταβλητή κλάσης mLinePaint, της γραμμής σχεδίασης δηλαδή για το τι χρώμα θα έχει και το τι φωτισμό. Για παράδειγμα με την γραμμή κώδικα

```
mLinePaint.setARGB(255, 255, 0, 0);
```

Η γραμμή σχεδίασης αποκτά το μπλε χρώμα.

Επίσης αρχικοποιεί τον πίνακα του buffer, ο οποίος κρατάει τα δεδομένα του buffer του δείγματος ήχου που ηχογραφήθηκε. Τέλος, θέτει το φόντο της εφαρμογής, που επιλέχθηκε να είναι μία μαύρη εικόνα.

Η μέθοδος **ChangeSensitivity()** είναι η μέθοδος που επιτρέπει να αλλάξει το μέγεθος της κλίμακας της κυματομορφής που σχεδιάζεται στην οθόνη σχεδίασης.

Η μέθοδος **doDraw**(), είναι η μέθοδος που ζωγραφίζει το φάσμα του ήχου. Παίρνει το μέγεθος του καμβά, δηλαδή της οθόνης σχεδίασης, το μέγεθος του κάθε buffer του δείγματος ήχου(ύψος και πλάτος) και με βάση την ChangeSensitivity() καθορίζει το μέγεθος της γραμμής.

Συνεχώς ηχογραφούμε όσο τρέχει η εφαρμογή. Για το λόγο αυτό είναι απαραίτητος ένας έλεγχος που να θέτει ποια είναι τα όρια της οθόνης σχεδίασης. Ξεκινάμε από το σημείο 0 της οθόνης και όσο δεν έχουμε φτάσει στην άλλη άκρη της οθόνης σχεδιάζουμε το φάσμα του ήχου.

```
while (StratX < width -1)
```

```
{
 int StartBaseY = mBuffer[(mBuffIndex - 1)] / scale;


 int StopBaseY = mBuffer[mBuffIndex] / scale;
 if (StartBaseY > height / 2)
 {
        StartBaseY = 2 + height / 2;
  int checkSize = height / 2;
  if (StopBaseY <= checkSize)
      return;
  StopBaseY = 2 + height / 2;
 }


  int StartY = StartBaseY + height / 2;
  int StopY = StopBaseY + height / 2;
  paramCanvas.drawLine(StratX, StartY, StratX +1, StopY, mLinePaint);
  cu1++;
  mBuffIndex++;
  StratX++;
  int checkSize_again = -1 * (height / 2);
  if (StopBaseY >= checkSize_again)
```

```
            continue;

        StopBaseY = -2 + -1 * (height / 2);

    }
```

Τέλος, έχουμε την κλάση MainActivity που είναι η κύρια κλάση της εφαρμογής. Η κλάση αυτή κληρονομεί την Activity, έτοιμη κλάση του Android.

Μια δραστηριότητα είναι μία ενιαία, εστιασμένη λειτουργία που μπορεί να επιτελέσει ο χρήστης. Σχεδόν όλες οι δραστηριότητες που αλληλεπιδρούν με το χρήστη. Η δραστηριότητα φροντίζει για τη δημιουργία ενός παραθύρου στο οποίο μπορεί να τοποθετηθεί το UI χρησιμοποιώντας την εντολή setContentView(View).

Κάθε Activity είναι υποχρεωμένη να υλοποιεί τις εξής μεθόδους:

1. **onCreate**(Bundle savedInstance) : Καλείται όταν η δραστηριότητα δημιουργείται για πρώτη φορά. Εδώ είναι το σημείο όπου μπορούμε να δημιουργήσουμε προβολές, να συνδέσουμε δεδομένα κλπ. Η onCreate δέχεται ως όρισμα ένα Bundle το οποίο είναι μία χαρτογράφηση από string τιμές που δείχνουν την κατάσταση του στιγμιότυπου. Αυτό που κάνει η onCreate είναι να σχεδιάζει το user interface που βλέπει ο χρήστης μέσω του layout της activity_main.xml, τη μαύρη οθόνη δηλαδή με τον τίτλο της εφαρμογής. Επίσης, δημιουργεί ένα στιγμιότυπο της κλάσης DrawSound.
2. **OnPause**(), Καλείται όταν το σύστημα είναι έτοιμο να ξεκινήσει την επιστροφή στην προηγούμενη δραστηριότητα. Αυτό συνήθως χρησιμοποιείται για να μη χαθούν μη αποθηκευμένες αλλαγές σε σταθερά δεδομένα, να σταματήσει κινούμενα σχέδια και άλλα πράγματα που μπορούν να καταναλώνουν CPU. Ακολουθείται είτε από onResume() εάν η δραστηριότητα επιστρέφει πίσω προς τα εμπρός, ή onStop() αν

γίνεται αόρατο στο χρήστη. Στη συγκεκριμένη περίπτωση, η onPause() καλεί την setRun() της DrawSound που περιγράφηκε παραπάνω και επίσης χειρίζεται και τα νήματα της εφαρμογής.

3. **OnRestart**(), η μέθοδος που καλείται μόλις σταματήσει η δραστηριότητα και προτού ξανάξεκινήσει. Ακολουθείται συνήθως από την onStart().

4. **OnStart(),** η μέθοδος που καλείται μόλις η δραστηριότητα γίνεται ορατή από το χρήστη

5. **onResume**(), η μέθοδος που καλείται μόλις η δραστηριότητα αρχίσει να αλληλεπιδρά με τον χρήστη. Στηριζόμενη σε κάποιες μεταβλητές απόφασης, η onResume() καλεί τη Restart() της DrawSound κάθε φορά που έχει συλλεχθεί δείγμα ήχου.

6. **OnStop**(), η μέθοδος που καλείται μόλις η δραστηριότητα παύει να είναι ορατή στον χρήστη.

Τέλος, η MainActivity έχει και μία μέθοδο ακόμη την **onRun**(). Η μέθοδος αυτή, φτιάχνει το drawThread και το στιγμιότυπο της κλάσης RecordSound και στη συνέχει μόλις ξεκινήσει η εφαρμογή εμφανίζει στο χρήστη ένα μήνυμα (Toast Message) που τον παροτρύνει να κάνει κάποιο θόρυβο.

```
Toast localToast = Toast.makeText(localContext, "Παρακαλώ κάνετε κάποιο θόρυβο..",
Toast.LENGTH_LONG);
```

Επίσης καλεί την startRecording() και startSampling().

Παρακάτω στο Παράρτημα Α παρατίθεται αναλυτικά ο κώδικας της εφαρμογής.

ΚΕΦΑΛΑΙΟ 4 : ΣΥΜΠΕΡΑΣΜΑΤΑ

Κατόπιν της έρευνας που διεξάχθηκε στα πλαίσια της παρούσας πτυχιακής εργασίας για την ανάπτυξη μίας εφαρμογής η οποία θα είναι σε θέση να ηχογραφεί ήχο από το φυσικό περιβάλλον της και θα απεικονίζει το φάσμα του ήχου, καταλήξαμε σε κάποιες παραδοχές.

Ο ήχος και τα σήματα φωνής είναι ένα πολύ ενδιαφέρον κομμάτι για την τεχνολογία που μπορεί να βρει εφαρμογές σε διάφορους τομείς όπως στην ιατρική, σε συστήματα ασφαλείας, σε στατιστικές έρευνες και διάφορα άλλα, όπως για παράδειγμα για έλεγχο του φάσματος ήχου κατά τη διάρκεια της ημέρας για την διεξαγωγή συμπερασμάτων που αφορούν το φάσμα του ήχου ανά ώρα.

Φυσικά, η εφαρμογή αναπτύχθηκε σε πλαίσια πτυχιακής εργασίας, οπότε ως αποτέλεσμα οι ικανότητές της είναι περιορισμένες. Μία μελλοντική προσθήκη θα μπορούσε να είναι η διάκριση εύφωνου ή άφωνου τμήματος ήχου.

ΚΕΦΑΛΑΙΟ 5 : ΒΙΒΛΙΟΓΡΑΦΙΑ

- http://el.wikipedia.org/wiki/Android/
- http://www.androidfreedownload.net/
- https://play.google.com/store/apps/details?id=radonsoft.net.spectralview
- http://el.wikipedia.org/wiki/%CE%89%CF%87%CE%BF%CF%82
- http://developer.android.com/reference/android/media/AudioRecord.html
- http://developer.android.com/reference/android/media/MediaRecorder.html
- https://www.eclipse.org/

  http://developer.android.com/tools/sdk/eclipse-adt.html
- https://developer.android.com/training/basics/firstapp/index.html?hl=el
- http://el.wikipedia.org/wiki/%CE%9D%CE%AE%CE%BC%CE%B1_(%CF%85%CF%80%CE%BF%CE%BB%CE%BF%CE%B3%CE%B9%CF%83%CF%84%CE%AD%CF%82)
- http://developer.android.com/reference/java/lang/Runnable.html
- http://developer.android.com/reference/android/app/Activity.html
- http://developer.android.com/reference/android/app/Activity.html
- http://xanthippi.ceid.upatras.gr/courses/mobile/2008_09/lpc.pdf

ΠΑΡΑΡΤΗΜΑ Α : Ο ΚΩΔΙΚΑΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

MainActivity

```
package visualizesound;


import android.os.Bundle;

import android.app.Activity;

import android.content.Context;

import android.util.Log;

import android.view.Display;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Toast;


public class MainActivity extends Activity {

        private DrawSound.CDrawThread drawThread;

        private DrawSound drawer;


         private OnClickListener listener;

         private Boolean start = Boolean.valueOf(false);

        private Boolean recording;
```

```
        private RecordSound sampler;


        @Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   drawer = (DrawSound) findViewById(R.id.drawer);
   start = Boolean.valueOf(false);




   while (true)
   {
     recording = Boolean.valueOf(false);
     run();
     System.out.println("mDrawThread NOT NULL");
     System.out.println("recorder NOT NULL");
     return;
   }
}
/**
 * Pause the visualizer when the app is paused
```

```
 */
@Override
protected void onPause()
{
  System.out.println("onpause");
  sampler.SetRun(Boolean.valueOf(false));
  drawThread.setRun(Boolean.valueOf(false));
  sampler.SetSleeping(Boolean.valueOf(true));
  drawThread.SetSleeping(Boolean.valueOf(true));
  Boolean.valueOf(false);
  super.onPause();
}
/**
 * Resters the visualizer when the app restarts
 */
@Override
protected void onRestart()
{
  start = Boolean.valueOf(true);
  System.out.println("onRestart");
  super.onRestart();
```

```
}

/**

 * Resume the visualizer when the app resumes

 */

@Override

protected void onResume()

{

 System.out.println("onResume");

 int i = 0;

 while (true)

 {

  if ((sampler.GetDead2().booleanValue()) && (drawer.GetDead2().booleanValue()))

  {

   System.out.println(sampler.GetDead2() + ", " + drawer.GetDead2());

   sampler.Restart();

   if (!start.booleanValue())

    drawer.Restart(Boolean.valueOf(true));

   sampler.SetSleeping(Boolean.valueOf(false));

   drawThread.SetSleeping(Boolean.valueOf(false));

   start = Boolean.valueOf(false);

   super.onResume();
```

```
  return;
}
try
{
  Thread.sleep(500L);
  System.out.println("Περιμένετε..");
  i++;
  if (!sampler.GetDead2().booleanValue())
    System.out.println("sampler not DEAD!!!");
  if (!drawer.GetDead2().booleanValue())
  {
    System.out.println("mDrawer not DEAD!!");
    drawer.SetRun(Boolean.valueOf(false));
  }
  if (i <= 4)
    continue;
  drawThread.SetDead2(Boolean.valueOf(true));
}
catch (InterruptedException localInterruptedException)
{
  localInterruptedException.printStackTrace();
```

```
    }
  }
}

@Override
protected void onStart()
{
  System.out.println("On Start");
  super.onStart();
}

@Override
protected void onStop()
{
  System.out.println("On Stop");
  super.onStop();
}

/**
 * Recives the buffert from the sampler
```

```
 * @param buffert
 */
public void setBuffer(short[] paramArrayOfShort)
{
  drawThread = drawer.getThread();
  drawThread.setBuffer(paramArrayOfShort);
}
/**
 * Called by OnCreate to get everything up and running
 */
public void run()
{
          try
          {
   if (drawThread == null)
   {
     drawThread = drawer.getThread();
   }
   if (sampler == null)
       sampler = new RecordSound(this);
  Context localContext = getApplicationContext();
```

```
    Display localDisplay = getWindowManager().getDefaultDisplay();

    Toast localToast = Toast.makeText(localContext, "Παρακαλώ κάνετε κάποιο θόρυβο..",
Toast.LENGTH_LONG);

    localToast.setGravity(48, 0, localDisplay.getHeight() / 8);

    localToast.show();

    drawer.setOnClickListener(listener);

    if (sampler != null){

    try {

                sampler.Init();

        } catch (Exception e) {

                // TODO Auto-generated catch block

                e.printStackTrace();

        }

    sampler.StartRecording();

    sampler.StartSampling();

    }

          } catch (NullPointerException e) {

                Log.e("Main_Run", "NullPointer: " + e.getMessage());

          }

   }

}
```

RecordSound

```
package visualizesound;



/**

 *  This is the sampler for the visualizer

 *  This collects the data the will be visualized

 *  @author Pontus Holmberg (EndLessMind)

 *  Email: the_mr_hb@hotmail.com

 **/



import android.media.AudioRecord;

import android.media.MediaPlayer;

import android.util.Log;

import visualizesound.*;



import java.io.PrintStream;



public class RecordSound

{

  private static final int SAMPPERSEC = 44100;

  private static short[] buffer;
```

```
private AudioRecord ar;

private int audioEncoding = 2;

private int buffersizebytes;

private int buflen;

private int channelConfiguration = 16;

private int mSamplesRead;

private Boolean m_bDead = Boolean.valueOf(false);

private Boolean m_bDead2 = Boolean.valueOf(true);

private Boolean m_bRun;

private Boolean m_bSleep = Boolean.valueOf(false);

private MainActivity m_ma;

private Thread recordingThread;



public RecordSound(MainActivity paramMainActivity)

{

  m_ma = paramMainActivity;

  m_bRun = Boolean.valueOf(false);

}



public Boolean GetDead2()

{
```

```
  return m_bDead2;
 }

 public Boolean GetSleep()
 {
  return m_bSleep;
 }

 /**
  * Prepares to collect audiodata.
 * @throws Exception
  */
 public void Init() throws Exception
 {
            try {
  if (!m_bRun)
  {
    ar = new AudioRecord(1, 44100, channelConfiguration, audioEncoding,
AudioRecord.getMinBufferSize(44100, channelConfiguration, audioEncoding));
    if (ar.getState() != 1)
    return;
```

```
   System.out.println("State initialized");
  }
            } catch (Exception e) {
                        Log.d("TE", e.getMessage());
                        throw new Exception();
            }
  while (true)
  {
   buffersizebytes = AudioRecord.getMinBufferSize(44100, channelConfiguration, audioEncoding);
   buffer = new short[buffersizebytes];
   m_bRun = Boolean.valueOf(true);
   System.out.println("State unitialized!!!");
   return;
  }
}

/**
 * Restarts the thread
 */
public void Restart()
{
```

```
while (true)
{
 if (m_bDead2.booleanValue())
 {
  m_bDead2 = Boolean.valueOf(false);
  if (m_bDead.booleanValue())
  {
   m_bDead = Boolean.valueOf(false);
   ar.stop();
   ar.release();
   try {
                    Init();
                } catch (Exception e) {
                    return;
                }
   StartRecording();
   StartSampling();
  }
  return;
 }
 try
```

```
    {
      Thread.sleep(1000L);
    }
    catch (InterruptedException localInterruptedException)
    {
      localInterruptedException.printStackTrace();
    }
  }
}


/**
 * Reads the data-bufferts
 */
public void Sample()
{
  mSamplesRead = ar.read(buffer, 0, buffersizebytes);
}



public void SetRun(Boolean paramBoolean)
{
```

```
  m_bRun = paramBoolean;
  if (m_bRun.booleanValue())
    StartRecording();
  while (true)
  {


    StopRecording();
    return;
  }
}


public void SetSleeping(Boolean paramBoolean)
{
  m_bSleep = paramBoolean;
}




public void StartRecording()
{
        if (ar == null) {
                try {
```

```
                    Init();
                } catch (Exception e) {
                    e.printStackTrace();
                }
                 StartRecording();
            } else {

                 ar.startRecording();
            }

}

/**
 * Collects audiodata and sends it back to the main activity
 */
public void StartSampling()
{
  recordingThread = new Thread()
  {
    public void run()
    {
```

```
    while (true)
    {
     if (!m_bRun.booleanValue())
     {
       m_bDead = Boolean.valueOf(true);
       m_bDead2 = Boolean.valueOf(true);
       return;
     }
     Sample();
     m_ma.setBuffer(RecordSound.buffer);
    }
   }
  };
  recordingThread.start();
}

public void StopRecording()
{
  ar.stop();
}
```

```
  public short[] getBuffer()
  {
    return buffer;
  }

}
```

DrawSound

```
package visualizesound;



/**

 *  This is the drawer for the visualizer

 *  @author Pontus Holmberg (EndLessMind)

 *  Email: the_mr_hb@hotmail.com

 **/



import android.content.Context;

import android.graphics.Bitmap;

import android.graphics.Bitmap.Config;

import android.graphics.Canvas;

import android.graphics.Paint;

import android.os.Handler;

import android.os.Message;

import android.util.AttributeSet;

import android.util.Log;

import android.view.SurfaceHolder;

import android.view.SurfaceHolder.Callback;

import android.view.SurfaceView;
```

```
import java.io.PrintStream;

import java.util.Arrays;



public class DrawSound extends SurfaceView



 implements SurfaceHolder.Callback



{



 private Context mContext;

 private CDrawThread mDrawThread;

 private SurfaceHolder mHolder;



 private Boolean isCreated = false;

 /**

  * This is where you instance the drawer

  * You relly don't need to care about the parameters, they are set in the xml-layout

  * @param Apply the baseContext of you current acitivty

  * @param AttributeSet

  */

 public DrawSound(Context paramContext, AttributeSet paramAttributeSet)
```

```
{
  super(paramContext, paramAttributeSet);
  System.out.println("CDrawer()");
  mHolder = getHolder();
  mContext = paramContext;
  mHolder.addCallback(this);
  mDrawThread = new CDrawThread(mHolder, paramContext, new Handler()
  {
    public void handleMessage(Message paramMessage)
    {
    }
  });
  mDrawThread.setName("" + System.currentTimeMillis());
  setFocusable(true);
}

public Boolean GetDead2()
{
  return mDrawThread.GetDead2();
}
```

```
/**
 * restarts the thread
 * @param Is the thread dead?
 */
public void Restart(Boolean paramBoolean)
{
            if (isCreated) {
 if (mDrawThread.GetDead2().booleanValue())
 {
   mDrawThread.SetDead2(Boolean.valueOf(false));
   if ((!paramBoolean.booleanValue()) || (!mDrawThread.GetDead().booleanValue()))

   mHolder = getHolder();
   mHolder.addCallback(this);
   System.out.println("Restart drawthread");
   mDrawThread = new CDrawThread(mHolder, mContext, new Handler()
   {
    public void handleMessage(Message paramMessage)
    {
    }
   });
```

```
 mDrawThread.setName("" + System.currentTimeMillis());
 mDrawThread.start();
 return;
}
Boolean No1,No2 = true;
while (true)
{
 while (No2 = true)
 {

  try
  {
   Thread.sleep(1000L);
   System.out.println("Restart");
   No2 = false;
   mDrawThread.SetDead2(Boolean.valueOf(true));
  }
  catch (InterruptedException localInterruptedException)
  {
   localInterruptedException.printStackTrace();
  }
```

```
    return;
   }

  if (!mDrawThread.GetDead().booleanValue())
    continue;
   mHolder = getHolder();
   mHolder.addCallback(this);
   System.out.println("Restart drawthread");
   mDrawThread = new CDrawThread(mHolder, mContext, new Handler()
   {
    public void handleMessage(Message paramMessage)
    {
    }
   });
   mDrawThread.setName("" +System.currentTimeMillis());
   mDrawThread.start();
   return;
  }
          }
}
```

```
public void SetRun(Boolean paramBoolean)
{
  mDrawThread.setRun(paramBoolean);
}

public CDrawThread getThread()
{
  return mDrawThread;
}

/**
 * Called when there's a change in the surface
 */
public void surfaceChanged(SurfaceHolder paramSurfaceHolder, int paramInt1, int paramInt2, int
paramInt3)
{
  mDrawThread.setSurfaceSize(paramInt2, paramInt3);
}

/**
 * Creates the surface
```

```
 */
public void surfaceCreated(SurfaceHolder paramSurfaceHolder)
{
  System.out.println("surfaceCreated");
  if (mDrawThread.getRun().booleanValue())
  {
    System.out.println("11111");
    isCreated = true;
    mDrawThread.start();

  }
  while (true)
  {
    System.out.println("22222");
    Restart(Boolean.valueOf(false));
    return;
  }
}

/**
 * Surface destroyd
```

```
 */

public void surfaceDestroyed(SurfaceHolder paramSurfaceHolder)

{

  int i = 1;

  while (true)

  {

    if (i == 0)

      return;

    try

    {

      mDrawThread.join();

      i = 0;

    }

    catch (InterruptedException localInterruptedException)

    {

    }

  }

}



/**

 * The Drawer Thread, subclass to cDrawer class
```

```
 * We want to keep most of this process in a background thread,
 * so the UI don't hang
 *  @author Pontus Holmberg (EndLessMind)
 *  Email: the_mr_hb@hotmail.com
 */
class CDrawThread extends Thread
{
  private Paint mBackPaint;
  private Bitmap mBackgroundImage;
  private short[] mBuffer;
  private int mCanvasHeight = 1;
  private int mCanvasWidth = 1;
  private Paint mLinePaint;
  private int mPaintCounter = 0;
  private SurfaceHolder mSurfaceHolder;
  private Boolean m_bDead = Boolean.valueOf(false);
  private Boolean m_bDead2 = Boolean.valueOf(true);
  private Boolean m_bRun = Boolean.valueOf(true);
  private Boolean m_bSleep = Boolean.valueOf(false);
  private int m_iScaler = 8;
  private int counter = 0;
```

```
/**
 * Instance the Thread
 * All the parameters i handled by the cDrawer class
 * @param paramContext
 * @param paramHandler
 * @param arg4
 */
public CDrawThread(SurfaceHolder paramContext, Context paramHandler, Handler arg4)
{
  mSurfaceHolder = paramContext;
  mLinePaint = new Paint();
  mLinePaint.setAntiAlias(true);
  mLinePaint.setARGB(255, 255, 0, 0);
  mLinePaint = new Paint();
  mLinePaint.setAntiAlias(true);
  mLinePaint.setARGB(255, 0, 0, 255);
  mBackPaint = new Paint();
  mBackPaint.setAntiAlias(true);
  mBackPaint.setARGB(255, 0, 0, 0);
  mBuffer = new short[2048];
  mBackgroundImage = Bitmap.createBitmap(1, 1, Bitmap.Config.ARGB_8888);
```

```
}

/**
 * Allow you to change the size of the waveform displayed on the screen
 * Or scale of you so will
 * @return returns a new scale value
 */
public int ChangeSensitivity()
{
  m_iScaler = (2 + m_iScaler);
  if (m_iScaler > 20)
    m_iScaler = 1;
  return m_iScaler;
}

public Boolean GetDead()
{
  return m_bDead;
}
```

```
public Boolean GetDead2()

{

  return m_bDead2;

}



public Boolean GetSleep()

{

  return m_bSleep;

}



public void SetDead2(Boolean paramBoolean)

{

  m_bDead2 = paramBoolean;

}



public void SetSleeping(Boolean paramBoolean)

{

  m_bSleep = paramBoolean;

}



/**
```

```
 * Calculate and draws the line
 * @param Canvas to draw on, handled by cDrawer class
 */
public void doDraw(Canvas paramCanvas)
{
  if (mCanvasHeight == 1)
    mCanvasHeight = paramCanvas.getHeight();
  paramCanvas.drawPaint(mBackPaint);
  /**
   * Set some base values as a starting point
   * This could be considerd as a part of the calculation process
   */
  int height = paramCanvas.getHeight();
  int BuffIndex = (mBuffer.length / 2 - paramCanvas.getWidth()) / 2;
  int width = paramCanvas.getWidth();
  int mBuffIndex = BuffIndex;
  int scale = height / m_iScaler;
  int StratX = 0;
  if (StratX >= width)
  {
    paramCanvas.save();
```

```
  return;
 }
        int cu1 = 0;
        /**
         * Here is where the real calculations is taken in to action
         * In this while loop, we calculate the start and stop points
         * for both X and Y
         *
         * The line is then drawer to the canvas with drawLine method
         */
while (StratX < width -1)
{
 int StartBaseY = mBuffer[(mBuffIndex - 1)] / scale;

 int StopBaseY = mBuffer[mBuffIndex] / scale;
 if (StartBaseY > height / 2)
 {
          StartBaseY = 2 + height / 2;
  int checkSize = height / 2;
  if (StopBaseY <= checkSize)
        return;
```

```
    StopBaseY = 2 + height / 2;
  }

    int StartY = StartBaseY + height / 2;
    int StopY = StopBaseY + height / 2;
    paramCanvas.drawLine(StratX, StartY, StratX +1, StopY, mLinePaint);
    cu1++;
    mBuffIndex++;
    StratX++;
    int checkSize_again = -1 * (height / 2);
    if (StopBaseY >= checkSize_again)
      continue;
    StopBaseY = -2 + -1 * (height / 2);
  }
}

public Boolean getRun()
{
  return m_bRun;
}
```

```
/**
 * Updated the Surface and redraws the new audio-data
 */
public void run()
{
  while (true)
  {
    if (!m_bRun.booleanValue())
    {
      m_bDead = Boolean.valueOf(true);
      m_bDead2 = Boolean.valueOf(true);
      System.out.println("Goodbye Drawthread");
      return;
    }
    Canvas localCanvas = null;
    try
    {
      localCanvas = mSurfaceHolder.lockCanvas(null);
      synchronized (mSurfaceHolder)
      {
          if (localCanvas != null)
```

```
        doDraw(localCanvas);



    }

    }

    finally

    {

      if (localCanvas != null)

        mSurfaceHolder.unlockCanvasAndPost(localCanvas);

    }

  }

}



public void setBuffer(short[] paramArrayOfShort)

{

  synchronized (mBuffer)

  {

    mBuffer = paramArrayOfShort;

    return;

  }

}
```

```
    public void setRun(Boolean paramBoolean)
    {
      m_bRun = paramBoolean;
    }

    public void setSurfaceSize(int paramInt1, int paramInt2)
    {
      synchronized (mSurfaceHolder)
      {
        mCanvasWidth = paramInt1;
        mCanvasHeight = paramInt2;
        mBackgroundImage = Bitmap.createScaledBitmap(mBackgroundImage, paramInt1, paramInt2,
true);
        return;
      }
    }
  }
}
```

Sleeper

```
package visualizesound;



/**

 *  This is the sleeper for the visualizer

 *  This allows the sampler to collect data before running.

 *  @author Pontus Holmberg (EndLessMind)

 *  Email: the_mr_hb@hotmail.com

 **/

import java.io.PrintStream;



public class Sleeper

  implements Runnable

{

  private Boolean done = Boolean.valueOf(false);

  private MainActivity m_ma;

  private RecordSound m_sampler;



  public Sleeper(MainActivity paramMainActivity, RecordSound paramCSampler)

  {

    m_ma = paramMainActivity;
```

```
  m_sampler = paramCSampler;
}

public void run()
{
  try {
                    m_sampler.Init();
          } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
          }
  while (true)
    try
    {
      Thread.sleep(1000L);
      System.out.println("Tick");
      continue;
    }
    catch (InterruptedException localInterruptedException)
    {
      localInterruptedException.printStackTrace();
```

```
    }

  }

}
```