ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι.
ΘΕΣΣΑΛΟΝΙΚΗΣ ΣΧΟΛΗ
ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Android και ανάπτυξη εφαρμογών σε mobile συσκευές

Του φοιτητή
Παπάρα Κωνσταντίνου
Αρ. Μητρώου 04/2558

Επιβλέπων καθηγητής
Σφέτσος Παναγιώτης

Θεσσαλονίκη 2013

ΠΡΟΛΟΓΟΣ

Το κείμενο αυτό αποτελεί τμήμα της πτυχιακής μου εργασίας με θέμα το «Android και ανάπτυξη εφαρμογών σε mobile συσκευές», σε συνδυασμό με την εφαρμογή MusicBee Remote. Η πτυχιακή και η εφαρμογή υλοποιηθήκαν στα πλαίσια των σπουδών μου στο τμήμα Πληροφορικής του ΑΤΕΙ Θεσσαλονίκης υπό την εποπτεία του καθηγητή κ. Παναγιώτη Σφέτσου.

Στόχος τις εργασίας ήταν η ανάπτυξη μιας μοντέρνας εφαρμογής για την πλατφόρμα Android με στόχο τα πρόσφατα API (15+), και χρήση βιβλιοθηκών όπως η ActionBarSherlock και Android Support Libraries v4, παρέχοντας μια ενιαία έκδοση για tablets και κινητά τηλέφωνα (smartphones), αξιοποιώντας εργαλεία τεχνικές και μεθόδους από τον χώρο της Java, του Android καθώς και από τον τομέα τις μηχανικής λογισμικού.

ΠΕΡΙΛΗΨΗ

Στο πρώτο κεφάλαιο παρουσιάζεται το IDE Intellij IDEA της Jetbrains το οποίο χρησιμοποιήθηκε κατά την ανάπτυξη της εφαρμογής και στη συνέχεια γίνεται μια παρουσίαση του Apache Maven build automation tool το οποίο χρησιμοποιήθηκε ως build system για το project σε συνδυασμό με το android maven plugin. Συγκεκριμένα γίνεται αναφορά στο Project Object Model (POM) το οποίο περιέχει τις ρυθμίσεις και πληροφορίες ενός project, στην τυπική δομή των φακέλων σε ένα Maven project καθώς και το πώς μπορεί να κάνει κάποιος build ένα project. Επίσης γίνεται μια σύντομη παρουσίαση του android maven plugin.

Στο επόμενο κεφάλαιο γίνεται παρουσίαση χαρακτηριστικών του Android SDK, τα οποία αποτελούν στοιχεία μιας μοντέρνας Android εφαρμογής, όπως τα Fragments, η ActionBar με τη χρήση της ActionBarSherlock, το Navigation Drawer, τα Notifications καθώς και ο ViewPager.

Στο τρίτο κεφάλαιο γίνεται μια παρουσίαση των βιβλιοθηκών που χρησιμοποιήθηκαν κατά την ανάπτυξη της εφαρμογής. Στα πλαίσια της παρουσίασης του RoboGuice framework γίνεται και μια παρουσίαση του σχεδιαστικού προτύπου Dependency Injection, ενώ στη συνέχεια παρουσιάζονται οι βιβλιοθήκες Otto Event Bus, DragSortListView, Crouton και Jackson.

Στο τελευταίο κεφάλαιο γίνεται παρουσίαση τμήματος του σχεδιασμού της εφαρμογής, καθώς και μια μικρή παρουσίαση του πρωτοκόλλου που χρησιμοποιείται για την απομακρυσμένη επικοινωνία.

ABSTRACT

In the first chapter there is a presentation of the IDE Intellij IDEA by Jetbrains that was used during the application development. The Apache Maven build automation system is also presented. Maven was used as a build system for the application in conjunction with android maven plugin. Specifically the Project Object Model (POM) that contains the information and configuration of a maven project is presented, along with the standard folder structure and information on the building process. There is also a reference to the maven android plugin.

In the next chapter the parts of the Android SDK used during the application development, most of them are considered parts of a modern Android application. These include the Fragments, the ActionBar through the usage of ActionBarSherlock, the Navigation Drawer, the Notifications, and the ViewPager.

In the third chapter the libraries used during the application development are presented. In the context of the RoboGuice framework presentation there is also a presentation of the Dependency Injection design pattern. Furthermore there is a presentation of the Otto event bus library, the DragSortListView, Crouton and Jackson.

Finally during the last chapter part of the application design is presented along with a small presentation of the protocol used for the remote communication.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Παναγιώτη Σφέτσο για την ανάθεση της πτυχιακής. Επιπλέον θα ήθελα να ευχαριστήσω την οικογένεια μου για την αμέριστη συμπαράσταση και την υποστήριξη τους.

Ακόμη θα ήθελα να ευχαριστήσω τους Ιορδάνη Γεωργιάδη για την ιδέα και το σχεδιασμό του λογότυπου της εφαρμογής και Αναστάσιο Παπάζογλου Χαλικιά για την πολύτιμη συμβολή του στο σχεδιασμό του user interface της εφαρμογής.

ΠΕΡΙΕΧΟΜΕΝΑ

Ευρετήριο Σχημάτων

Ευρετήριο Πινάκων

ΕΙΣΑΓΩΓΗ

Το Android είναι μια συνεχώς αναπτυσσόμενη και ραγδαία εξελισσόμενη πλατφόρμα. Από την εμφάνιση του, μέχρι σήμερα έχει καταφέρει να αποκτήσει ένα αρκετά μεγάλο κομμάτι τις αγοράς. Μόνο το δεύτερο τρίμηνο του 2013 πουλήθηκαν 177,898 χιλιάδες smartphones με Android αναλογώντας στο 79% των συνολικών πωλήσεων smartphones.

Στόχος αυτής της πτυχιακής είναι η δημιουργία μιας μοντέρνας εφαρμογής για την πλατφόρμα η οποία χρησιμοποιεί τα πιο πρόσφατα χαρακτηριστικά από το Android SDK όπως τα Fragments, ακολουθώντας όσο το δυνατό τις σχεδιαστικές προτάσεις όπως αυτές παρουσιάζονται στο Android Design (σχεδιαστικά guidelines για τους developers), κάνοντας χρήση σχεδιαστικών design patterns όπως είναι η ActionBar, το Navigation Drawer τα Notifications κ.α. Η υποστήριξη παλαιότερων συσκευών γίνεται με τη χρήση της βιβλιοθήκης υποστήριξης (Android Support Libraries).

Κατά την ανάπτυξη της εφαρμογής έγινε χρήση του Apache Maven ως build system και έγινε η χρήση του git ως version control κατά την ανάπτυξη της εφαρμογής. Ο κώδικας της εφαρμογής είναι διαθέσιμος στο Github.

Η εφαρμογή που αναπτύχθηκε λειτουργεί για την απομακρυσμένη διαχείριση ενός προγράμματος μουσικής (MusicBee). Χρησιμοποιεί TCP sockets για να πετύχει απομακρυσμένη σύνδεση, με ένα plugin το οποίο λειτουργεί ως socket server (το plugin έχει άμεση επικοινωνία με το API του player). Η επικοινωνία βασίζεται σε ένα αυτοσχέδιο πρωτόκολλο το οποίο μεταφέρει τα μηνύματα μέσω του socket χρησιμοποιώντας το JSON format.

Η χρήση της εφαρμογής όπως αναφέρθηκε επιτρέπει στο χρήστη την απομακρυσμένη διαχείριση του player. Ο χρήστης μπορεί να το τρέχον κομμάτι (πληροφορίες όπως τίτλο, καλλιτέχνη, άλμπουμ κ.α.) μαζί με το εξώφυλλο του στην οθόνη της Android συσκευής του. Μπορεί επίσης να αλλάξει την ένταση του player η να μετακινήσει τη θέση αναπαραγωγής στο κομμάτι που παίζει βλέποντας παράλληλα την τρέχουσα θέση και την διάρκεια του κομματιού. Επιπλέον δίνεται η δυνατότητα στο χρήστη να βαθμολογήσει το τρέχον κομμάτι.

Η εφαρμογή παρέχει επίσης στο χρήστη τη δυνατότητα να δει την λίστα αναπαραγωγής και να διαλέξει κάποιο συγκεκριμένο κομμάτι για αναπαραγωγή, ή απλά να ψάξει για αυτό με βάση τον τίτλο του. Επιπλέον δίνεται η δυνατότητα στο χρήστη να αναδιοργανώσει τη λίστα του με τη χρήση drag-n-drop, καθώς και να αφαιρέσει κομμάτι από αυτή.

Ακόμη παρέχεται η δυνατότητα στο χρήστη να ψάξει στην μουσική του βιβλιοθήκη με βάση το είδος μουσικής, τον καλλιτέχνη, το άλμπουμ ή τον τίτλο του κομματιού και να το προσθέσει στη λίστα αναπαραγωγής του για μελλοντική ή άμεση αναπαραγωγή.

Η εφαρμογή διαθέτει επίσης ένα μηχανισμό εύκολης ρύθμισης για Wi-Fi δίκτυα ο οποίος κάνει χρήση της τεχνικής multicast με το πρωτόκολλο UDP.

Για την επίτευξη της λειτουργίας της εφαρμογής έγινε η χρήση ενός αριθμού από βιβλιοθήκες οι οποίες παρουσιάζονται στην πορεία.

ΚΕΦΑΛΑΙΟ 1

Εργαλεία ανάπτυξης

Εισαγωγή

Στο κεφάλαιο αυτό γίνεται μια σύντομη παρουσίαση του IDE που χρησιμοποιήθηκε κατά την ανάπτυξη της εφαρμογής, το οποίο είναι το Intellij IDEA της εταιρίας Jetbrains.

Στη συνέχεια γίνεται παρουσίαση του συστήματος αυτοματοποίησης Apache Maven, χαρακτηριστικών του καθώς και του τρόπου λειτουργίας του. Παράλληλα παρουσιάζεται το Project Object Model το οποίο χρησιμοποιείται για τον ορισμό των ρυθμίσεων γύρω από ένα maven project. Το Apache Maven χρησιμοποιείται για την παραγωγή του τελικού εκτελέσιμου της Android εφαρμογής.

Παράλληλα παρουσιάζονται πληροφορίες σχετικά με το πρόσθετο android-maven-plugin το οποίο χρειάζεται για να παρέχει στο Maven την δυνατότητα να δημιουργεί και να πακετάρει την εφαρμογή στην κατάλληλη μορφή για την πλατφόρμα.

Επιπλέον γίνεται μια μικρή παρουσίαση του βοηθητικού εργαλείου Maven Android SDK Deployer το οποίο είναι απαραίτητο στην περίπτωση που η εφαρμογή μας στοχεύει τις ποιο πρόσφατες εκδόσεις της πλατφόρμας Android.

Τέλος γίνεται μια μικρή αναφορά σε αντίστοιχα συστήματα που χρησιμοποιούνται για τον ίδιο σκοπό.

1.1 Intellij IDEA

Κατά την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε το IDE Intellij IDEA της εταιρίας Jetbrains. Πρόκειται για ένα Java IDE το οποίο είναι διαθέσιμο σε δύο εκδόσεις, μια Community Edition η οποία είναι δωρεάν διαθέσιμη ως λογισμικό ανοιχτού κώδικα, καθώς και μια κλειστή εμπορική έκδοση. Κατά την ανάπτυξη έγινε κυρίως χρήση των EAP[1] εκδόσεων 12 και 13. Αξίζει να σημειωθεί ότι η Community Edition της έκδοσης 13 αποτελεί τη βάση του νέου IDE Android Studio το οποίο αναπτύσσεται σε συνεργασία με τη Google και προορίζεται ως εναλλακτικό περιβάλλον ανάπτυξης του Eclipse με το ADT.

Από τα κύρια χαρακτηριστικά του IDEA είναι ο Smart Code Completion μηχανισμός ο οποίος παρέχει προτάσεις στο χρήστη καθώς αυτός πληκτρολογεί, αναγνωρίζοντας και λαμβάνοντας υπόψιν το τρέχον context καθώς και το τρέχον scope, ώστε να παρέχει όσο το δυνατόν πιο ακριβείς προτάσεις..

Παράλληλα χρησιμοποιεί ένα μηχανισμό ο οποίος πραγματοποιεί σε πραγματικό χρόνο (on the fly) ανάλυση κώδικα (static code analysis) παρουσιάζοντας στο χρήστη προειδοποιήσεις και πιθανά σφάλματα καθώς αυτός πληκτρολογεί. Ακόμη ο μηχανισμός έχει τη δυνατότητα να ανακαλύψει πιθανά

[1] Οι EAP ή αλλιώς early access preview εκδόσεις είναι δοκιμαστικές pre-release εκδόσεις οι οποίες παρέχονται στους προγραμματιστές της κοινότητας για δοκιμή κατά την διάρκεια της ανάπτυξης μιας νέας έκδοσης, αυτό σημαίνει ότι πολλές φορές μπορεί να υπάρχουν σημαντικά προβλήματα τα οποία δεν θα υπήρχαν σε μια stable version.

προβλήματα στον κώδικα και να προτείνει πιθανές λύσεις, δίνοντας στον προγραμματιστή τη δυνατότητα να τις εφαρμόσει άμεσα αν το επιθυμεί. Επιπλέον παρέχετε η δυνατότητα για την πραγματοποίηση ελέγχων είτε σε ολόκληρο στο project, ή σε τμήματα αυτού εμφανίζοντας προειδοποιήσεις και πιθανά σφάλματα ομαδοποιημένα κατά κατηγορία και τύπο.

Ένα από τα κεντρικά χαρακτηριστικά του IDEA, και ένα από τα σημαντικότερα είναι το πλήθος των δυνατότητων που παρέχει για την αναδόμηση (refactoring) κώδικα. Ενδεικτικά μερικά παραδείγματα των δυνατοτήτων αυτών αποτελούν η δυνατότητα εξαγωγής class (Extract Class) η οποία επιτρέπει την εξαγωγή ενός υποσυνόλου των πεδίων και των μεθόδων σε μια νέα class με πολύ απλό τρόπο, ή η δυνατότητα τροποποίησης (Type Migration) τύπου που επιτρέπει την αυτόματη αλλαγή του τύπου ενός μέλους, για παράδειγμα από integer σε string, αλλάζοντας επίσης όλες τις εξαρτώμενες χρήσεις κατά τη ροή των δεδομένων, όπως τοπικές μεταβλητές, τύπους επιστροφής κ.α. σε όλο το project, καθώς και την εύκολη και αυτοματοποιημένη μετατροπή μεταξύ arrays και collections. Το IDEA παρέχει ένα μεγάλο πλήθος ενσωματωμένων λειτουργιών για την αναδόμηση κώδικα (refactoring) με στόχο να διευκολύνει τον προγραμματιστή την διαδικασία βελτίωσης του κώδικα, αυτοματοποιώντας ένα πλήθος λειτουργιών οι οποίες σε κάθε άλλη περίπτωση θα απαιτούσαν την τροποποίηση πλήθους αρχείων ένα προς ένα.

Μεταξύ των σημαντικών χαρακτηριστικών του IDEA είναι η υποστήριξη που παρέχει για ένα πλήθος από τα πιο συνηθισμένα build automation tool για την ανάπτυξη Java εφαρμογών. Ενδεικτικά υποστηρίζονται εργαλεία όπως τα Maven, Ant, Gradle και Gant. Πιο συγκεκριμένα ο χρήστης έχει τη δυνατότητα να ορίσει την εκτέλεση συγκεκριμένων ενεργειών του maven, με συγκεκριμένη σειρά ώστε αυτές να εκτελούνται κάθε φορά που πατά το κουμπί build του IDE.

Επιπλέον το IDEA παρέχει μια πολύ καλή ενσωμάτωση μεσώ ενός ενιαίου interface για ένα πλήθος version control systems στα οποία συμπεριλαμβάνονται τα Git, Subversion και Mercurial μεταξύ άλλων. Στον ενσωματωμένο file explorer του IDEA ο προγραμματιστής έχει την δυνατότητα να δει ποια αρχεία έχουν τροποποιηθεί από το προηγούμενο commit, ποια αρχεία έχουν προστεθεί η διαγραφεί από το version control system (vcs) καθώς και ποια είναι τα αρχεία των οποίων οι αλλαγές δεν παρακολουθούνται από το vcs. Τέλος μέσω του interface ο χρήστης έχει δυνατότητα να κάνει commit τα αρχεία και να πραγματοποιήσει ενέργειες που επιτρέπονται από το vcs.

Η community edition με την ultimate (commercial) έχουν ένα πλήθος διαφορές, μια από τις σημαντικότερες είναι η υποστήριξη ανάπτυξης σε διαφορετικές γλώσσες. Η community edition παρέχει κυρίως υποστήριξη σε γλώσσες που βασίζονται στο JVM όπως για παράδειγμα τη Java, τη Scala, τη Groovy κ.α. Σε αντίθεση η ultimate υποστηρίζει ένα μεγάλο πλήθος από γλώσσες συμπεριλαμβανομένων των Ruby, Python, PHP, JavaScript κ.α. Η πλατφόρμα IDEA αποτελεί τη βάση άλλων εξειδικευμένων IDE της Jetbrains όπως είναι τα RubyMine, Pycharm, PhpStorm κλπ. Όπως αναφέρθηκε η community edition υποστηρίζει και την ανάπτυξη εφαρμογών Android και αποτελεί την βάση για το νέο IDE Android Studio που αναπτύσσεται σε συνεργασία με τη Google.

Άλλη μια σημαντική διαφοροποίηση της ultimate έκδοσης είναι η ενσωματωμένη υποστήριξη για την ανάπτυξη εφαρμογών (enterprise και web) με

την χρήση συγκεκριμένων frameworks. Στα υποστηριζόμενα frameworks συμπεριλαμβάνονται το Spring και το Play. Επιπλέον παρέχεται υποστήριξη και για την ανάπτυξη εφαρμογών με τη χρήση της πλατφόρμας Java Enterprise Edition καθώς και η υποστήριξη για ένα πλήθος από διαφορετικούς application servers όπως για παράδειγμα ο Apache Tomcat.

1.2 Maven

Η ονομασία του εργαλείου προέρχεται από την Γίντις (Yiddish)[2] λέξη και έχει την σημασία του ανθρώπου που καταλαβαίνει, με βάση την γνώση που έχει συγκεντρώσει. Ο maven είναι ένας έμπιστος ειδικός σε ένα συγκεκριμένο πεδίο ο οποίος αναζητά να περάσει την γνώση του σε άλλους.

Το maven ξεκίνησε αρχικά σαν μια προσπάθεια για την απλοποίηση της build process στο project Jakarta Turbine, και στόχος του ήταν να παρέχει ένα τυποποιημένο τρόπο για την διαδικασία, ένα ξεκάθαρο ορισμό των τμημάτων από τα οποία αποτελείται, παρέχοντας παράλληλα ένα εύκολο τρόπο για την δημοσίευση πληροφοριών του project καθώς και ένα τρόπο ώστε να γίνεται εύκολα επαναχρησιμοποίηση βιβλιοθηκών (JAR) σε διαφορετικά project.

Το maven χρησιμοποιεί ένα XML αρχείο για να περιγράψει το project, τις βιβλιοθήκες ή άλλες εξωτερικές μονάδες και συστατικά από τα οποία εξαρτάται, την σειρά με την οποία πρέπει να δημιουργηθούν τα διάφορα στοιχεία, τους φακέλους και τα απαραίτητα πρόσθετα.

Το maven μεταφορτώνει δυναμικά τις απαιτούμενες Java βιβλιοθήκες καθώς και τα απαραίτητα πρόσθετα από ένα ή περισσότερα repositories όπως το Maven Central Repository και τα αποθηκεύει τοπικά. Το τοπικό repository ενημερώνεται επίσης με αντικείμενα τα οποία έχουν δημιουργηθεί από τοπικά project.

1.3 Project Object Model

Το Project Object Model (POM) παρέχει όλες τις απαραίτητες διαμορφώσεις (configuration) για ένα project. Το POM είναι βρίσκεται αποθηκευμένο σε ένα αρχείο που ονομάζεται `pom.xml`. Μέσα στις πληροφορίες που περιέχει περιλαμβάνονται το όνομα του ιδιοκτήτη, το όνομα του project, οι εξαρτήσεις του από άλλα project (για παράδειγμα βιβλιοθήκες) κ.α.

Παρέχεται επίσης η δυνατότητα στο χρήστη να παραμετροποιήσει συγκεκριμένες φάσεις της διαδικασίας παραγωγής, οι οποίες υλοποιούνται με τη χρήση προσθέτων. Για παράδειγμα, δίνεται η δυνατότητα να ρυθμιστεί το πρόσθετο μεταγλώττισης (Maven Compiler Plugin) ώστε να χρησιμοποιήσει την έκδοση 1.5 της Java ή να οριστεί το πακετάρισμα (packaging) του project ακόμη και στην περίπτωση που κάποιο από τα unit test αποτύχει.

Τα μεγάλα project μπορούν να χωριστούν σε περισσότερα τμήματα ή υπό project, από τα οποία το καθένα έχει το δικό του POM. Αυτό δίνει την δυνατότητα στον για ένα κεντρικό POM από το οποίο είναι εφικτό να μεταγλωττιστούν όλα τα επιμέρους τμήματα με τη χρήση μιας μόνο εντολής. Τα POM έχουν τη δυνατότητα

[2] Yiddish είναι μια γλώσσα μέλος των Υψηλών Γερμανικών Γλωσσών που πηγάζει από τους Ασκενάζι και αποτελεί ένα μείγμα από διαφορετικές γερμανικές διαλέκτους, με λέξεις που πηγάζουν από τα Εβραίικα, τα Αραμαϊκά, τα Σλάβικα κ.α.

να κληρονομήσουν την διαμόρφωση (configuration) τους από άλλα POM και όλα κληρονομούν από το Super POM το οποίο παρέχει προεπιλεγμένες ρυθμίσεις, όπως τον προεπιλεγμένο φάκελο για τον πηγαίο κώδικα κ.α.

1.3.1. Βασικά Στοιχεία

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

  <!-- application info-->
    <groupId>com.kelsos.mbrc</groupId>
    <artifactId>mbrc</artifactId>
    <version>0.9.8-SNAPSHOT</version>
  <packaging>apk</packaging>
  <name>MusicBee Remote</name>

  <properties>
   <android.version>4.3_r1</android.version>
  </properties>

  <dependencies>
   <dependency>
            <groupId>android</groupId>
            <artifactId>android</artifactId>
            <version>${android.version}</version>
       <scope>provided</scope>
   </dependency>
  </dependencies>
</project>
```

Στο παραπάνω παράδειγμα βλέπουμε ένα τμήμα από το Project Object Model. Το tag `<modelVersion>` περιέχει την τιμή 4.0.0, αυτή είναι η μόνη υποστηριζόμενη τιμή - έκδοση του POM για τις εκδόσεις 2 και 3 του maven και η τιμή είναι απαραίτητη.

Στη συνέχεια έχουμε το tag `<groupId>`, αυτό είναι ένα μοναδικό αναγνωριστικό το σε ένα οργανισμό ή κάποιο project. Τα group ID δεν χρησιμοποιούν απαραίτητα την σήμανση με τελείες και ούτε είναι απαραίτητο να ανταποκρίνεται στην δομή των πακέτων που περιέχει το project, παρόλα αυτά είναι μια καλή πρακτική που καλό θα ήταν να ακολουθείτε.

Το `<artifactId>` αντιπροσωπεύει γενικά το όνομα με το οποίο είναι γνωστή η εφαρμογή. Σε συνδυασμό με το `groupId` το `<artifactId>` καθορίζει τη θέση του αντικειμένου μέσα στο repository. Για παράδειγμα η θέση της εφαρμογής μας στο repository είναι `$M2_REPO`[3]`/com/kelsos/mbrc/mbrc`.

Το tag version αντιπροσωπεύει την έκδοση του project στην οποία αναφέρεται ο πηγαίος κώδικας. Η έκδοση χρησιμοποιείται στη δομή των φακέλων των repositories για να ξεχωρίσει τα αντικείμενα διαφορετικών εκδόσεων. Για παράδειγμα η έκδοση 0.9.5 βρίσκεται στο φάκελο

[3] Στα περισσότερα συστήματα τα τοπικά repositories βρίσκονται στο φάκελο .m2/repository ο οποίος είναι διαθέσιμος μέσα στον home φάκελο του χρήστη.

`$M2_REPO/com/kelsos/mbrc/mbrc/0.9.5/`, ενώ η έκδοση 0.9.7 είναι αντίστοιχα στον υπό φάκελο 0.9.7. Όσον αφορά τις εκδόσεις το maven ακολουθεί μια συγκεκριμένη σήμανση για τα αντικείμενα των εκδόσεων που βρίσκονται υπό ανάπτυξη, η σήμανση αυτή επιτυγχάνεται προσθέτοντας στο τέλος της έκδοσης τη λέξη SNAPSHOT, π.χ. 0.9.7-SNAPSHOT. Σε αντίθεση με τις κανονικές εκδόσεις το maven ελέγχει για καινούργια snapshots στο απομακρυσμένο repository κάθε φορά που γίνεται build του project. Αυτό γίνεται ιδιαίτερα χρήσιμο στις περιπτώσεις μεγάλων ομάδων που δουλεύουν ταυτόχρονα σε διαφορετικά τμήματα ενός μεγάλο project ώστε να κάνουν τις τυχόν αλλαγές τους άμεσα διαθέσιμες σε στις υπόλοιπες ομάδες που δουλεύουν σε διαφορετικά τμήματα.

Επιπλέον έχουμε διαθέσιμο το tag `<packaging>` το οποίο περιγράφει τη μορφή του αντικειμένου η οποία στην περίπτωση μας είναι apk. Πρόκειται για τη μορφή που θα έχει το αρχείο που θα δημιουργηθεί μετά το πέρας του build process, στην περίπτωση του Android έχουμε apk αρχεία που αντιπροσωπεύουν μια εφαρμογή Android, μια Java βιβλιοθήκη θα είχε για παράδειγμα jar στο `<packaging>`.

Στο παράδειγμα μας βλέπουμε επίσης πως ορίζεται ένα dependency, που στην περίπτωση μας είναι το Android SDK για την έκδοση 4.3 του Android OS, κάθε dependency δηλώνεται κάτω από το tag `<dependencies>`, ουσιαστικά κάθε `<dependency>` είναι μια βιβλιοθήκη η κάποιο framework το οποίο χρησιμοποιείται μέσα στο maven project. Θα παρατηρήσουμε ότι το tag για το `<scope>` του `<dependency>` έχει την τιμή provided, που σημαίνει ότι το συγκεκριμένο dependency παρέχεται κατά τον χρόνο της εκτέλεσης από την πλατφόρμα και δεν είναι απαραίτητο να συμπεριληφθεί στο πακέτο κατά την διαδικασία της μεταγλώττισης.

Στις δυνατές τιμές για το tag περιλαμβάνονται επίσης οι εξής.

- **compile**, είναι η προεπιλεγμένη τιμή και χρησιμοποιείται αν δεν έχει δηλωθεί ρητά κάτι διαφορετικό. Οι εξαρτήσεις που έχουν δηλωθεί ως compile είναι διαθέσιμες σε όλα τα classpath και μεταδίδονται σε άλλα projects που χρησιμοποιούν το συγκεκριμένο project.
- **runtime**, είναι απαραίτητη κατά την εκτέλεση αλλά όχι κατά τη μεταγλώττιση.
- **test**, δεν είναι απαραίτητη για την κανονική χρήση της εφαρμογής αλλά μόνο κατά το στάδιο τις μεταγλώττισης και εκτέλεσης των unit test της εφαρμογής.

Τέλος ένα από βασικά τμήματα που απαιτούνται για την κατανόηση του POM είναι οι ιδιότητες. Οι ιδιότητες χρησιμοποιούνται ως σταθερές για την διατήρηση τιμών (placeholders). Οι τιμές των ιδιοτήτων είναι διαθέσιμες σε οποιοδήποτε σημείο του POM και η πρόσβαση σε αυτές μπορεί να επιτευχθεί απλά με τη χρήση της σήμανσης `${x}`, όπου το x συμβολίζει το όνομα της ιδιότητας, για παράδειγμα η χρήση του property `${android.version}`, θα τοποθετήσει την τιμή 4.3_r1 στο tag `<version>` όπως φαίνεται στο παράδειγμα πιο πάνω.

1.3.2. Build Settings

Πέρα από τα βασικά στοιχείο του POM έχουμε το tag <build>. Το <build> είναι εννοιολογικά διαχωρισμένο σε δύο ενότητες. Υπάρχει ένας BaseBuild τύπος ο οποίος περιέχει ένα σύνολο από στοιχεία κοινά και στα δύο (το build στοιχείο που βρίσκεται στο ανώτερο επίπεδο, και αυτό που βρίσκεται κάτω από το στοιχείο profiles).

Στο <build> tag του ανώτερου επιπέδου θα βρούμε τα plugins κάτω από το tag <plugin> το οποίο περιέχει το στοιχείο <configuration>. Το στοιχείο <configuration> περιέχει ρυθμίσεις που είναι ειδικά για το συγκεκριμένο πρόσθετο.

Υπάρχει ακόμη το στοιχείο <pluginManagement> το οποίο συναντάται κυρίως μαζί με τα πρόσθετα και περιέχει plugin elements όπως και το build tag με τη διαφορά ότι οι ρυθμίσεις του προορίζονται για τα project builds που κληρονομούν από αυτό και όχι για το ίδιο.

1.3.3. Πληροφορίες και περιβάλλον

Κάποια επιπλέον tags που έχουν να κάνουν με πληροφορίες για το project είναι τα ακόλουθα, το <licenses> στο οποίο δηλώνεται το license του κώδικα, το <organization> στο οποίο δηλώνεται ο οργανισμός ο οποίος αναπτύσσει το λογισμικό, το tag <developers> που περιέχει πληροφορίες για τους ανθρώπους που εμπλέκονται στην ανάπτυξη του λογισμικού (συνήθως αυτούς με τους οποίους θα πρέπει να επικοινωνήσει κάποιος αν παρουσιαστεί η ανάγκη).

Επίσης υπάρχουν tags διαθέσιμα για τις ρυθμίσεις περιβάλλοντος, όπως για παράδειγμα το <issueManagement> που ορίζει το issue tracking system και χρησιμοποιείται συνήθως για τη δημιουργία documentation, το <ciManagement> το οποίο περιλαμβάνει ρυθμίσεις για το Continuous Integration σύστημα. Έχουμε ακόμη το <scm> tag το οποίο περιέχει πληροφορίες για το code version control system (για παράδειγμα Git, Subversion κ.α.) το οποίο χρησιμοποιεί το project. Το <scm> tag χρησιμοποιείται από το maven release plugin κατά την προετοιμασία (prepare) του release, όπου αυτοματοποιεί την διαδικασία διαχείρισης του version control κάνοντας αυτόματα κάποιες διαδικασίες που σε άλλη περίπτωση θα απαιτούσαν την επέμβαση του προγραμματιστή.

1.3.4. Profiles

Επίσης το POM παρέχει τη δυνατότητα για την ύπαρξη διαφορετικών προφίλ. Τα προφίλ δίνουν την δυνατότητα να αλλάζουν οι ρυθμίσεις ανάλογα με το περιβάλλον στο οποίο γίνεται build η εφαρμογή. Για παράδειγμα δίνεται η δυνατότητα ύπαρξης ξεχωριστών profile για release και debug builds τα οποία πραγματοποιούν ένα διαφορετικό σύνολο ενεργειών.

```
<profile>
  <id>standard</id>
  <activation>
   <activeByDefault>true</activeByDefault>
  </activation>
</profile>
```

Στην περίπτωση μας έχουμε το standard profile που είναι το προεπιλεγμένο και, ενεργοποιείται αυτόματα αν δεν οριστεί ρητά η χρήση κάποιου άλλου προφίλ. Το standard προφίλ χρησιμοποιεί το debug keystore για την ψηφιακή υπογραφή του apk. Αντίθετα το release profile περιέχει ρυθμίσεις ώστε να χρησιμοποιεί διαφορετικό keystore το οποίο περιέχει το ιδιωτικό κρυπτογραφικό κλειδί με το οποίο θα υπογραφεί το apk προτού ανέβει στο Google Play.

Για να αποφευχθεί η δημοσιεύση των προσωπικών κλειδιών και κωδικών, ιδιαίτερα όταν πρόκειται για λογισμικό ανοιχτού κώδικα το οποίο είναι διαθέσιμο στο ευρύ κοινό μέσω για παράδειγμα του Github, υπάρχει η δυνατότητα να δηλώσουμε τις τιμές ως properties στο `settings.xml` αρχείο το οποίο βρίσκεται στο φάκελο `$HOME/.m2/` και στο configuration του `maven-jarsigner-plugin` να χρησιμοποιήσουμε τα properties αυτά αντί για τις πραγματικές τιμές.

```
<keystore>${sign.keystore}</keystore>
<alias>${sign.alias}</alias>
<storepass>${sign.storepass}</storepass>
<keypass>${sign.keypass}</keypass>
```

1.4 Δομή

Ένα maven project τείνει να ακολουθεί μια συγκεκριμένη δομή όσον αφορά τους φακέλους του project. Έχουμε αρχικά τον κεντρικό φάκελο οποίος περιέχει τους υπό φακέλους και το `pom.xml`. Μαζί του συνήθως συμπεριλαμβάνονται το αρχείο που περιλαμβάνει το license του project μαζί με κάποιο πιθανό readme αρχείο καθώς και αναφορές (notices) που μπορεί να απαιτούνται από τις βιβλιοθήκες.

Η τυπική δομή ενός maven project περιλαμβάνει επίσης τον φάκελο `src` ο οποίος περιέχει με τη σειρά τους δύο ακόμη φακέλους τους `main` και `test`. Ο `main` περιλαμβάνει όλα εκείνα τα στοιχεία που απαιτούνται για το build του project οπότε ο `src/main/java` περιλαμβάνει τον πηγαίο κώδικα του project και ο `src/main/resources` όλα εκείνα τα resources που απαιτούνται. Εδώ το Android maven project διαφοροποιείται (ακολουθώντας το τυπικό μιας Android εφαρμογής) μιας και τα resources του είναι διαθέσιμα στο φάκελο res ο οποίος βρίσκεται στον κεντρικό φάκελο και περιλαμβάνει όλα τα resources, όπως layouts, drawables, γραφικά κ.α. Από την άλλη ο φάκελος test περιλαμβάνει ότι απαιτείται για το unit testing της εφαρμογής με το `src/test/java` να είναι ο προορισμός για τον πηγαίο κώδικα των unit test της εφαρμογής.

Είναι σημαντικό να αναφερθεί ότι από τη συγκεκριμένη εφαρμογή απουσιάζει ο `test` φάκελος μιας και δεν περιέχει unit tests.

1.5 Build

Το build ενός maven project μπορεί να επιτευχθεί μεσώ της γραμμής εντολών με την εκτέλεση της εντολής `mvn clean install`. Οι ενέργειες που μπορούν να εκτελεστούν από το maven (εντολή mvn) ονομάζονται στόχοι (goals), ο στόχος clean έχει ως σκοπό να καθαρίσει τα στοιχεία που έχουν διατηρηθεί από το προηγούμενο build και ο στόχος install να δημιουργήσει ένα νέο apk και να το εγκαταστήσει στο τοπικό repository. Για να εγκαταστήσουμε το apk σε μια συνδεμένη συσκευή αρκεί να τρέξουμε την εντολή `mvn android:deploy`.

Καλό είναι να σημειωθεί ότι λόγο της ενσωμάτωσης που παρέχει το IDEA για το maven έχουμε τα δυνατότητα να δημιουργήσουμε configurations τα οποία εκτελούν maven goals μέσω του IDE.

Για να δημιουργήσουμε ένα release build μπορούμε κάνουμε χρήση της εντολής `mvn clean install –Prelease`, η παράμετρος release ενημερώνει το maven ώστε αυτό να ενεργοποιήσει το release profile και όχι το standard.

Ένας ακόμη καλύτερος τρόπος για να προετοιμαστεί ένα release build είναι με τη χρήση του maven release plugin.

```
mvn release:prepare -DignoreSnapshots=true -DpreparationGoals='clean install -
Prelease --batch-mode
```

Η παράμετρος release:prepare κάνει την προετοιμασία του release, η παράμετρος `ignoreSnapshots` που χρησιμοποιείται ενημερώνει το maven ότι θα πρέπει να συνεχίσει το build ακόμη και στην περίπτωση που το project εξαρτάται από snapshot βιβλιοθήκες. Στην περίπτωση μας η βιβλιοθήκη `DragSortListView` είναι snapshot το οποίο σημαίνει ότι χωρίς την παραπάνω παράμετρο θα ήταν αδύνατο να πραγματοποιηθεί ένα release build. Στη συνέχεια έχουμε τα `preparationGoals` αυτά είναι οι στόχοι που πρέπει να επιτευχθούν κατά την προετοιμασία. Η παράμετρος `--batch mode` αφήνει το maven να πραγματοποιήσει κάποιες αλλαγές μόνο (όπως ο ορισμός της επόμενη έκδοσης υπό ανάπτυξη) του χωρίς την εισαγωγή του χρήστη.

Τρέχοντας την εντολή το maven θα κάνει προετοιμασία για το release αλλάζοντας την έκδοση αφαιρώντας το snapshot από την έκδοση, στη συνέχεια θα κάνει commit τις αλλαγές στο repository (source version control) δηλαδή στην περίπτωση μας στο git repository του project και θα δημιουργήσει ένα tag του κώδικα για τη συγκεκριμένη έκδοση. Έπειτα θα κάνει checkout το tag και θα τροποποιήσει την έκδοση του POM ώστε αυτή να αντικατοπτρίζει την καινούργια υπό ανάπτυξη έκδοση πραγματοποιώντας ακόμη ένα commit. Στο ενδιάμεσο της διαδικασίας αυτής θα δημιουργηθεί ένα release build όπως ορίζεται από τα `preparationGoals` το οποίο θα τοποθετηθεί στο τοπικό maven repository. Το release build είναι έτοιμο διάθεση μέσω του Google Play Store.

1.6 Maven και Android

Η δυνατότητα για την ανάπτυξη Android εφαρμογών με την χρήση του maven γίνεται εφικτή χάρη στο πρόσθετο android-maven-plugin. Το plugin είναι λογισμικό ανοιχτού κώδικα και είναι διαθέσιμο κάτω από το Apache License v2. Ο πηγαίος κώδικας του είναι διαθέσιμος μέσω του Github. Υπάρχει ένα αριθμός από προγραμματιστές που έχουν συνεισφέρει στην ανάπτυξη του πρόσθετου αλλά οι 2 σημαντικότεροι είναι οι Manfred Moser και Hugo Josefson.

Το Plugin βοηθά ώστε να αυτοματοποιηθεί πλήρως η διαδικασία προετοιμασίας ενός apk ώστε αυτό να διατεθεί στους τελικούς χρήστες. Η προεπιλογή του plugin είναι να υπογράφει το apk package με το debug keystore, η παράμετρος `<sign><debug>false<debug><sign>` χρησιμοποιείται για να

απενεργοποιήσει αυτή τη συμπεριφορά. Στην συνέχει ορίζεται ότι θα πρέπει να τρέξει το zipalign[4] καθώς και από πιο αρχείο θα διαβάσει και σε ποιο θα γράψει.

Στη συνέχεια ορίζονται οι αλλαγές που θα πραγματοποιήσει το plugin στο `AndroidManifest.xml`, το `<debuggable>false</debuggable>` ορίζει ότι το plugin θα τροποποιήσει το attribute `android:debuggable` του `<application>` tag που βρίσκεται στο manifest αλλάζοντας του την τιμή σε false. Το `<versionCodeAutoIncrement>` ορίζει ότι το πρέπει να αυξηθεί αυτόματα κατά ένα το attribute `android:versionCode` του manifest το οποίο ορίζει την εσωτερική έκδοση της εφαρμογής που χρησιμοποιείται για να ξεχωρίζει τα διαφορετικά apk. Το `<proguard><skip>false</skip></proguard>` ορίζει ότι θα πρέπει να εκτελεστεί το ProGuard[5] για το συγκεκριμένο build. Τέλος το τμήμα των `<executions>` πότε θα πρέπει να εκτελεστούν συγκεκριμένα goals του plugin, στο παράδειγμα μας βλέπουμε ότι το manifest update συμβαίνει κατά την φάση της επεξεργασίας των resources, ενώ το zipalign κατά τη φάση του packaging. Το συγκεκριμένο configuration αποτελεί τμήμα του release profile και το αποτέλεσμα του θα είναι ένα apk (Android Application) το οποίο θα έχει περάσει από όλα τα στάδια προετοιμασίας που απαιτούνται για να ανέβει στο Google Play Store συμπεριλαμβανομένης της υπογραφής με το προσωπικό κρυπτογραφικό κλειδί.

```
<plugin>
  <groupId>com.jayway.maven.plugins.android.generation2</groupId>
  <artifactId>android-maven-plugin</artifactId>
  <version>3.6.1</version>
  <inherited>true</inherited>
  <configuration>
   <sign>
       <debug>false</debug>
   </sign>
   <zipalign>
       <skip>false</skip>
       <verbose>true</verbose>
       <inputApk>${project.build.directory}/${project.artifactId}.apk</inputApk>
       <outputApk>${project.build.directory}/${project.artifactId}-signed-
aligned.apk
       </outputApk>
   </zipalign>
   <manifest>
       <debuggable>false</debuggable>
       <versionCodeAutoIncrement>true</versionCodeAutoIncrement>
   </manifest>
   <proguard>
       <skip>false</skip>
   </proguard>
  </configuration>
  <executions>
   <execution>
       <id>manifestUpdate</id>
       <phase>process-resources</phase>
```

[4] Το zipalign είναι ένα εργαλείο που χρησιμοποιείται για την ευθυγράμμιση ενός archive, το οποίο παρέχει σημαντικές βελτιστοποιήσεις σε ένα αρχείο εφαρμογής του Android (.apk). Ο σκοπός του είναι να διασφαλιστεί ότι όλα τα μη συμπιεσμένα δεδομένα ξεκινούν με μια ιδιαίτερη ευθυγράμμιση σε σχέση με την αρχή του αρχείου.

[5] Το ProGuard είναι ένα εργαλείο το οποίο συρρικνώνει, βελτιστοποιεί και κάνει obfuscate τον πηγαίο κώδικα αφαιρώντας τμήματα που δεν χρησιμοποιούνται, καθώς και μετονομάζοντας classes, πεδία και μεθόδους χρησιμοποιώντας σημασιολογικά ασαφή ονόματα.

```
      <goals>
            <goal>manifest-update</goal>
      </goals>
  </execution>
  <execution>
      <id>alignApk</id>
      <phase>package</phase>
      <goals>
            <goal>zipalign</goal>
      </goals>
  </execution>
 </executions>
</plugin>
```

Τέλος αξίζει να γίνει μια αναφορά στο εργαλείο Maven Android SDK Deployer. Η χρήση του εργαλείου είναι ώστε να παρέχει την δυνατότητα ανάπτυξης Android Maven εφαρμογών στοχεύοντας τις πολύ πρόσφατες εκδόσεις του Android. Το εργαλείο είναι απαραίτητο λόγο της μη έγκαιρης ενημέρωσης των κεντρικών Maven Repositories με τις τελευταίες εκδόσεις. Το εργαλείο αναλαμβάνει να εγκαταστήσει στα τοπικά repositories (`~/.m2/repository/`) τις απαραίτητες βιβλιοθήκες που απαιτούνται για την ανάπτυξη Android εφαρμογών. Το SDK Deployer χρησιμοποιεί ένα διαφορετικό `<groupId>` από αυτό που χρησιμοποιείται από τα επίσημα Android artifacts ώστε να αποφευχθούν πιθανά προβλήματα.

Για να χρησιμοποιηθεί το εργαλείο απαιτείται να είναι εγκατεστημένη η πιο πρόσφατη έκδοση του Android SDK. Το εργαλείο βασίζεται στο maven και η χρήστη του είναι απλή, για παράδειγμα για την εγκατάσταση της έκδοσης 4.3 αρκεί να εκτελεστεί η εντολή `mvn install -P 4.3`.

Επίλογος

Το Intellij IDEA είναι ένα υπέροχο IDE για την ανάπτυξη εφαρμογών Java και Android, άλλωστε όπως αναφέρθηκε αποτελεί τη βάση για το νέο IDE Android Studio που αναπτύσσεται σε συνεργασία Google για την ανάπτυξη Android εφαρμογών.

Το Apache Maven είναι ένα πολύ χρήσιμο εργαλείο για την ανάπτυξη Java και Android εφαρμογών, επιτρέποντας ένα ενιαίο μηχανισμό για build είτε μέσα από κάποιο IDE όπως το IDEA και το Eclipse (μέσω του m2eclipse) είτε από το command line, είτε μέσω κάποιου Continuous Integration συστήματος όπως για παράδειγμα το Jenkins. Παρόλα αυτά δεν είναι χωρίς κριτική αφού υπάρχουν και αυτοί που θεωρούν ότι το maven δεν λειτουργεί όπως θα έπρεπε και δεν είναι αρκετά εφικτό να διορθωθεί λόγο σχεδιαστικών προβλημάτων (Broyer, 2013). Άλλα build systems που χρησιμοποιούνται είναι είτε ενσωματωμένα στο IDE, το Eclipse (ADT) έχει δικό του build system, και το ίδιο ισχύει και για το IDEA. Έπειτα υπάρχει και το Apache Ant το οποίο μπορεί να παρέχει και δυνατότητα build μέσω του command line καθώς και μέσω των IDE. Το Ant είναι αρκετά ευέλικτο και θεωρείται πολύ καλή επιλογής για μικρά project τα οποία δεν χρησιμοποιούν μεγάλο αριθμό από βιβλιοθήκες, σε αντίθεση με το maven το οποίο είναι συνήθως καταλληλότερο για μεγαλύτερα project τα οποία απαιτούν και ένα μεγαλύτερο

αριθμό από διαφορετικές βιβλιοθήκες οι οποίες πιθανόν να αναπτύσσονται και από διαφορετικές ομάδες.

Τέλος αξίζει να αναφερθεί ότι η ομάδα ανάπτυξης του Android δημιουργεί ένα νέο build σύστημα το οποίο έχει στόχο να αντικαταστήσει ότι χρησιμοποιείται μέχρι τώρα (ADT, Ant κλπ.). Το νέο σύστημα έχει ως βάση στο Gradle και σκοπός του είναι να παρέχει ένα επεκτάσιμο και παραμετροποιήσιμο σύστημα το οποίο χρησιμοποιείται παντού (σε οποιοδήποτε IDE, μέσω command line, σε κάποιο continuous integration server κλπ.). Το Gradle χρησιμοποιεί μια γλώσσα ειδικού σκοπού (domain specific language) για να δηλώσει ότι είναι απαραίτητο για το build μιας android εφαρμογής. Επιπλέον παρέχει ευελιξία στην διαχείριση βιβλιοθηκών, παρέχοντας στο χρήστη τη δυνατότητα να χρησιμοποιήσει είτε απομακρυσμένα maven repositories είτε τοπικά αποθηκεμένες βιβλιοθήκες. Προς το παρόν φαίνεται ότι υπάρχουν κάποιοι περιορισμοί, κυρίως λόγω του ότι βρίσκεται ακόμη υπό ανάπτυξη, αλλά φαίνεται ένα αρκετά ενδιαφέρον σύστημα. Είναι σημαντικό ότι η ανάπτυξη γίνεται λαμβάνοντας υπόψιν τα προβλήματα, και τα παράπονα που έχουν εκφράσει οι Android developers για το build process των εφαρμογών.

Στο επόμενο κεφάλαιο θα δούμε τμήματα του Android SDK τα οποία χρησιμοποιήθηκαν κατά την ανάπτυξη της εφαρμογής και αποτελούν σημαντικό κομμάτι μιας μοντέρνας Android εφαρμογής.

ΚΕΦΑΛΑΙΟ 2

ANDROID SDK

Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζονται τμήματα του Android SDK τα οποία χρησιμοποιήθηκαν κατά την ανάπτυξη της εφαρμογής. Πολλά από αυτά τα τμήματα έχουν προστεθεί σε πρόσφατες εκδόσεις του Android όμως υπάρχει η δυνατότητα να γίνει χρήση τους σε παλαιότερες συσκευές με την χρήση του Android Support Library. Στις περισσότερες περιπτώσεις ο τρόπος με τον οποίο παρουσιάζονται τα τμήματα του SDK αφορά τις support και όχι τις native εκδόσεις τους, αν και οι διαφοροποιήσεις είναι συνήθως μικρές.

Τα τμήματα που παρουσιάζονται αφορούν τα Fragments, την ActionBar, το Navigation Drawer, τα Notifications και τέλος ο ViewPager, μαζί με τη βιβλιοθήκη ViewPagerIndicator. Στην περίπτωση της ActionBar η υλοποίηση παρουσιάζετε με τη χρήση της βιβλιοθήκης ActionBarSherlock, μιας και η αντίστοιχη ActionBarCompat δεν ήταν διαθέσιμη κατά την ανάπτυξη της εφαρμογής, αλλά προστέθηκε πολύ αργότερα.

2.1 Fragments

Ένα fragment αποτελεί μια συμπεριφορά ή ένα τμήμα της διεπαφής χρήστη μέσα σε ένα Activity. Ο προγραμματιστής έχει τη δυνατότητα να συνδυάσει πολλαπλά fragments σε activity ή να χρησιμοποιήσει ένα fragment σε πολλά activities. Ένα fragment έχει το δικό του κύκλο ζωής και δέχεται τα δικά του γεγονότα εισόδου, τα οποία μπορούν να προστεθούν ή να αφαιρεθούν όσο εκτελείται το activity.

Ένα fragment πρέπει πάντα να είναι ενσωματωμένο σε ένα activity και ο κύκλος ζωής του εξαρτάται άμεσα από τον κύκλο ζωής του activity που το φιλοξενεί. Για παράδειγμα όταν διακόπτεται προσωρινά η εκτέλεση ενός activity τότε διακόπτεται και η εκτέλεση όλων των fragments που περιέχονται σε αυτό. Όταν ένα activity καταστρέφεται το ίδιο ισχύει και για τα fragments του.

Παρόλα αυτά όσο ένα activity εκτελείται υπάρχει η δυνατότητα γίνει τροποποίηση του κάθε fragment ξεχωριστά, καθώς και να προστεθούν ή να αφαιρεθούν δυναμικά. Όταν πραγματοποιείται μια τέτοια συναλλαγή με fragments παρέχεται η δυνατότητα αυτή να προστεθεί στο back stack. Η διαχείριση του back stack γίνεται από το activity και κάθε εγγραφή σε αυτό αναπαριστά μια μετάβαση από ένα fragment σε ένα άλλο η οποία μπορεί να χρησιμοποιηθεί για αναιρέσει τις όποιες αλλαγές με τη χρήση του back button.

Όταν ένα fragment προστίθεται στο layout ενός activity ζει σε ένα `ViewGroup` μέσα στην ιεραρχία των views του activity και ορίζει το δικό του view layout. Ένα fragment μπορεί να χρησιμοποιηθεί είτε στατικά δηλώνοντας το στο xml layout του activity, είτε προσθέτοντας το δυναμικά σε κάποιο ήδη υπάρχουν `ViewGroup` κατά το χρόνο εκτέλεσης.

```
<fragment
  android:layout_width="wrap_content"
  android:layout_height="@dimen/mini_control_height"
       android:name="com.kelsos.mbrc.ui.fragments.MiniControlFragment"
  tools:layout="@layout/ui_fragment_mini_control"/>
```

Το παραπάνω αποτελεί τμήμα του `ui_fragment_nowplaying.xml` layout και είναι παράδειγμα ενός fragment που έχει δηλωθεί μέσα στο xml layout. Η αναφορά στο ποιο fragment θα πρέπει να δημιουργηθεί γίνεται γράφοντας στο attribute `android:name` το πλήρες package μέσα στο οποίο βρίσκεται το η fragment class που μας ενδιαφέρει. Στην περίπτωση μας το `MiniControlFragment` αποτελεί ένα τμήμα λειτουργικότητας που επαναλαμβάνεται σε περισσότερα από ένα fragments.

Στο `MainFragmentActivity` μπορούμε να δούμε ένα παράδειγμα fragment το οποίο δημιουργείται και προστίθεται δυναμικά κατά το χρόνο εκτέλεσης. Αν κοιτάξουμε το layout `ui_main_container.xml` θα δούμε ότι υπάρχει ένα `FrameLayout` το οποίο έχει id `frame_container`. Θα παρατηρήσουμε ότι το `FrameLayout` δεν περιέχει τίποτα εσωτερικά, υπάρχει μόνο ως placeholder για να φιλοξενεί τα fragments που προστίθενται δυναμικά στο activity.

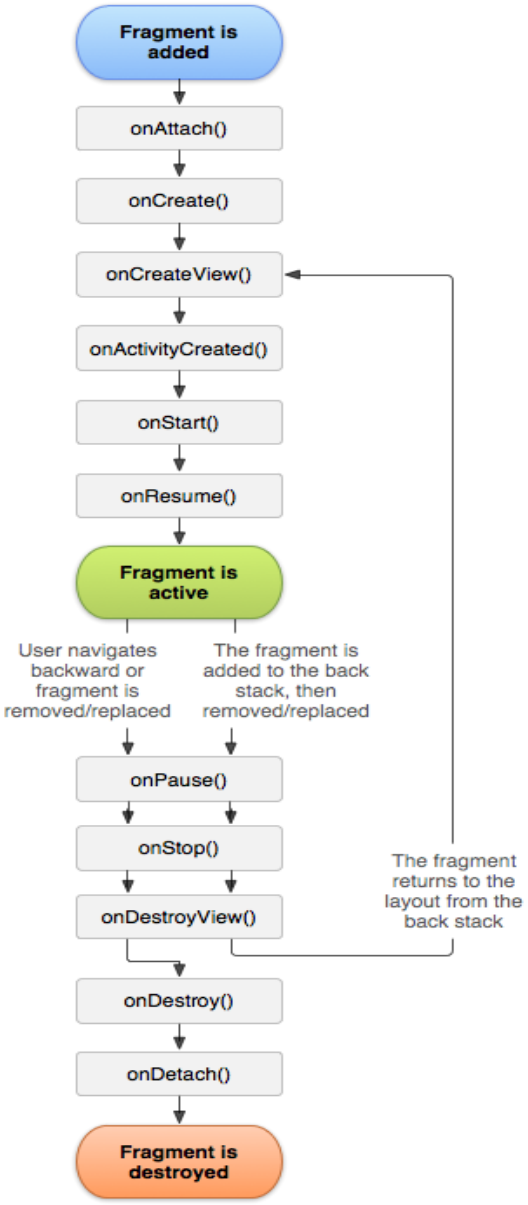
```
<FrameLayout
       android:id="@+id/fragment_container"
       android:layout_width="match_parent"
       android:layout_height="match_parent"/>
```

Όπως μπορούμε να δούμε στο `MainFragmentActivity` στην μέθοδο `onCreate()` έχουμε τη δυναμική δημιουργία ενός fragment. Αρχικά δημιουργούμε ένα νέο αντικείμενο της `MainFragment` class περνάμε ότι τυχόν παραμέτρους υπάρχουν και δημιουργούμε ένα νέο transaction. Λόγω του ότι τα fragments προστεθήκαν αρχικά στην έκδοση 3.0 του Android (API level 11) και δεν είναι διαθέσιμα σε παλαιότερες εκδόσεις, για να επιτευχθεί η χρήση των fragments σε παλαιότερες εκδόσεις απαιτείται η χρήση της Android Support Library v4. Η μέθοδος `getSupportFragmentManager()` είναι τμήμα της class `Fragment` (`android.support.v4.app.fragment`) του support library και αναλαμβάνει την δυναμική αλλαγή των fragments. Αφού δημιουργήσουμε ένα νέο transaction ορίζουμε το id του στοιχείου στο οποίο θα γίνει η αντικατάσταση καθώς και το νέο fragment (αντικείμενο) στην κλήση της μεθόδου replace ώστε να αντικαταστήσουμε το περιεχόμενο του fragment_container με το instance της `MainFragment`. Για να ολοκληρωθεί το transaction και να αντικατασταθεί το fragment πρέπει να κληθεί η μέθοδος commit για το instance του transaction.

```
MainFragment mFragment = new MainFragment();
mFragment.setArguments(getIntent().getExtras());

FragmentTransaction fragmentTransaction =
getSupportFragmentManager().beginTransaction();
fragmentTransaction.replace(R.id.fragment_container, mFragment,
"main_fragment");
fragmentTransaction.commit();
```

Η φιλοσοφία πίσω από το σχεδιασμό των fragments είναι να παρέχουν τη δυνατότητα για περισσότερο δυναμικά και ευέλικτα user interfaces, ιδιαίτερα σε μεγαλύτερες οθόνες όπως τα tablets. Επειδή τα tablets έχουν μεγαλύτερες οθόνες από τα τηλέφωνα υπάρχει μεγαλύτερος χώρος για συνδυασμό στοιχείων. Τα fragments παρέχουν ένα εύκολο τρόπο για να επιτευχθεί αυτό. Διαχωρίζοντας το layout ενός activity σε fragments παρέχεται η δυνατότητα να τροποποιήσουμε την εμφάνιση της εφαρμογής κατά το χρόνο εκτέλεσης και να διατηρήσουμε αυτές τις αλλαγές ώστε να μπορούμε να τις αναιρέσουμε.

Για να δημιουργηθεί ένα fragment απαιτείται η δημιουργία μιας υποκλάσης της κλάσης `Fragment`. Ο κώδικας της `Fragment` μοιάζει αρκετά με αυτό τις κλάσης `Activity`. Περιέχει callback μεθόδους όπως `onCreate()`, `onStart()`, `onPause()`, και `onStop()`.



Εικόνα 1. Κύκλος ζωής Fragment

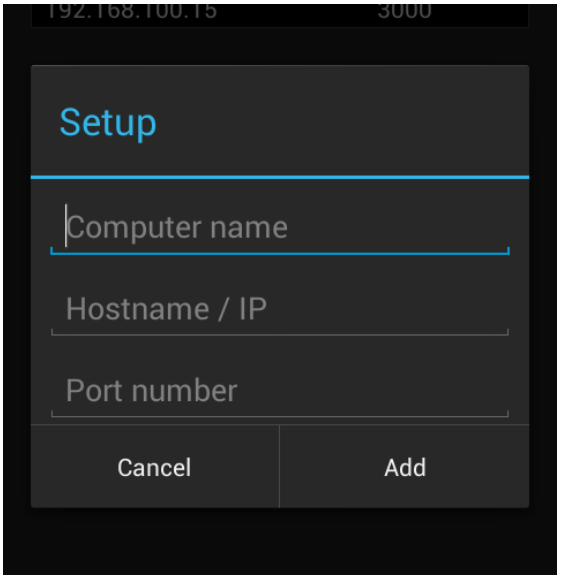
Συνήθως πρέπει να υλοποιηθούν τουλάχιστον κάποιες από τις μεθόδους του κύκλου ζωής της εφαρμογής όπως οι `onCreate()`, η οποία καλείται από το σύστημα όταν δημιουργείται το fragment. Στην υλοποίηση της `onCreate` γίνεται αρχικοποίηση των σημαντικών στοιχείων του fragment τα οποία πρέπει να διατηρηθούν κατά τη διάρκεια του κύκλου ζωής της εφαρμογής.

Στη συνέχεια έχουμε την μέθοδο `onCreateView` η οποία καλείται από το σύστημα τι στιγμή που το fragment εμφανίζει το user interface του για πρώτη φορά. Για να δημιουργηθεί το user interface του fragment απαιτείται η μέθοδος να επιστρέψει ένα View αντικείμενο το οποίο είναι το κομβικό σημείο του layout του fragment. Η μέθοδος έχει τη δυνατότητα να επιστρέψει null τιμή στην περίπτωση που το fragment δεν παρέχει γραφικό τμήμα.

Ακόμη είναι η μέθοδος `onPause` η οποία καλείται όταν ο χρήστης εγκαταλείπει το fragment (γιατί για παράδειγμα έχει μεταβεί σε κάποιο άλλο) η κλήση της μεθόδου από το σύστημα δεν σημαίνει απαραίτητα ότι το fragment θα καταστραφεί. Σε αυτό το σημείο καλό θα ήταν να αποθηκεύονται οποιεσδήποτε αλλαγές πρέπει να διατηρούνται αφότου κλείσει η εφαρμογή μιας και δεν υπάρχει καμία εγγύηση ότι ο χρήστης θα επιστρέψει στο fragment.

DialogFragment

Υπάρχουν διαθέσιμες επίσης μερικές subclasses της κλάσης fragment οι οποίες προορίζονται για συγκεκριμένες χρήσεις όπως είναι για παράδειγμα το `DialogFragment` που χρησιμοποιείται για τη δημιουργία παραθύρων διαλόγου στην εφαρμογή. Ένα τέτοιο παράδειγμα είναι η κλάση `SettingsDialogDragment` η οποία υλοποιεί ένα παράθυρο διαλόγου για την εισαγωγή ρυθμίσεων από το χρήστη.



Εικόνα 2. SettingsDialogFragment

Το `DialogFragment` αποτελείται από δύο τμήματα, το ένα είναι το xml layout (`ui_dialog_settings.xml`) το οποίο περιγράφει τη δομή του layout όμοια με τα layouts που χρησιμοποιούμε για activities και fragments και το άλλο είναι η κλάση που κληρονομεί τη `DialogFragment` και περιέχει την λογική (κώδικα) του παραθύρου διαλόγου.

```
@Override public Dialog onCreateDialog(Bundle savedInstanceState) {
  AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
  LayoutInflater inflater = getActivity().getLayoutInflater();

  builder.setView(inflater
.inflate(R.layout.ui_dialog_settings, null))
       .setTitle(R.string.dialog_application_setup_title)
       .setPositiveButton(R.string.settings_dialog_add,
new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialogInterface, int i) {

       }
   })        .setNegativeButton(R.string.dialog_application_setup_negative,
new DialogInterface.OnClickListener() {
       @Override
       public void onClick(DialogInterface dialogInterface, int i) {
   SettingsDialogFragment.this.getDialog().cancel();
       }
   });
  return builder.create();
}
```

Αντί να γίνει χρήση μόνο της `onCreateView` για την δημιουργία τις ιεραρχίας των στοιχείων μέσα στο παράθυρο διαλόγου, υλοποιούμε τη μέθοδο `onCreateDialog` για να δημιουργήσουμε ένα `AlertDialog`. Το `AlertDialog` είναι μια sub class της `Dialog` η οποία μπορεί να δείξει από ένα μέχρι τρία κουμπιά μαζί με το περιεχόμενο του διαλόγου.

Όπως μπορούμε να δούμε στο παραπάνω τμήμα κώδικα αρχικά δημιουργούμε αντικείμενα του `AlertDialog.Builder` καθώς και του `LayoutInflater`. Κάνουμε inflate το layout `ui_dialog_settings` και το ορίζουμε ως view για το Dialog που πρόκειται να εμφανίσουμε. Στη συνέχεια ορίζουμε τον τίτλο που θα εμφανιστεί στο πάνω μέρος του διαλόγου με τη μέθοδο `setTitle()`. Η `setPositiveButton` ορίζει το κείμενο για το κουμπί θετικής απάντησης (στην περίπτωση μας το κουμπί Add καθώς και την μέθοδο που θα διαχειριστεί το πάτημα του πλήκτρου. Η μέθοδος `setNegativeButton` διαχειρίζεται την αρνητική απάντηση (κουμπί cancel) και όπως θα δούμε η `OnClickListener` μέθοδος της ακυρώνει και συνεπώς κλείνει το παράθυρο διαλόγου. Στο τέλος της μεθόδου καλούμε την μέθοδο create η οποία δημιουργεί ένα διάλογο με τις παραμέτρους που δώσαμε στον builder.

```
SettingsDialogFragment settingsDialog = new SettingsDialogFragment();
Bundle args = new Bundle();
args.putInt("index", -1);
settingsDialog.setArguments(args);
settingsDialog.show(getSupportFragmentManager(), "settings_dialog");
```

Έπειτα μπορούμε να πάμε στο `ConnectionManagerActivity` όπου υπάρχει ο listener για το κουμπί που διαχειρίζεται το άνοιγμα του παραθύρου διαλόγου. Δημιουργούμε το νέο αντικείμενο τις κλάσης `SettingsDialogFragment` και στην συνέχεια αφού περάσουμε τις διάφορες παραμέτρους και καλούμε την μέθοδο show δίνοντας τις ως όρισμα τον `SupportFragmentManager`.

Λόγο της χρήσης του το SettingsDialogFragment πρέπει να περάσει τις ρυθμίσεις πίσω στο ConnectionManagerActivity. Για το λόγο αυτό υπάρχει το interface SettingsDialogListener το οποίο παρέχει τη μέθοδο onDialogPositiveClick το οποίο υλοποιεί η κλάση ConnectionManagerActivity.

```
public interface SettingsDialogListener {
      public void onDialogPositiveClick(DialogFragment dialog,
ConnectionSettings settings);
}
```

Ακόμη έχουμε την onAttach μέθοδο η οποία καλείται κατά την αρχική σύνδεση του fragment με το activity και συνδέει την υλοποίηση του SettingsDialogListener με το SettingsDialogFragment.

```
@Override public void onAttach(Activity activity) {
  super.onAttach(activity);

  try {
   mListener = (SettingsDialogListener) activity;
  } catch (ClassCastException e) {
   throw new ClassCastException(activity.toString() + " must implement
SettingsDialogListener");
  }
}
```

Επιπλέον αναφορικά στις διαθέσιμες subclasses της Fragment έχουμε τη ListFragment η οποία χρησιμοποιείται για να δείξει μια λίστα από αντικείμενα τα οποία τα διαχειρίζεται ένας adapter. Η ListFragment μοιάζει αρκετά με τη ListActivity και παρέχει αρκετές μεθόδους για τη διαχείριση ενός list view. Επίσης υπάρχει και η class PreferenceFragment η οποία ασχολείται με τη διαχείριση των αντικειμένων που έχουν να κάνουν με τις ρυθμίσεις χρήστη.

2.2 ActionBar

Η action bar είναι ένα χαρακτηριστικό του παραθύρου το οποίο προσδιορίζει τη θέση του χρήστη στη εφαρμογή και παρέχει ένα αριθμό από ενέργειες τις οποίες μπορεί να πραγματοποιήσει ο χρήστης. Παρέχει στο χρήστη μερικές βασικές λειτουργίες.

Εικόνα 3. ActionBar η οποία περιέχει [1] το εικονίδιο της εφαρμογής, [2] ένα action item, και [3] το κουμπί υπερχείλισης (overflow button)

- Παρέχει χώρο για την ταυτότητα της εφαρμογής και πληροφορίες για της θέση του χρήστη στην εφαρμογή.
- Παρουσιάζει σημαντικές ενέργειες σε προφανή θέση με ένα προβλεπόμενο τρόπο (όπως για παράδειγμα την αναζήτηση)
- Υποστηρίζει περιήγηση και αλλαγή παρουσιάσεων (με την χρήση των tabs ή drop-down lists)

Η ActionBar είναι διαθέσιμη από την έκδοση 3.0 του Android (API 11) και μετά. Για την χρήση της action bar στην έκδοση 2 του android υπάρχουν δύο επιλογές. Η μια επιλογή η οποία είναι και η πιο πρόσφατη είναι η προσθήκη του ActionBar ως τμήμα του Android Support Library (Ιούλιος 2013). Η action bar είναι τμήμα του v7 appcompat library και παρέχει υποστήριξη για τη χρήση της action bar σε συσκευές που τρέχουν εκδόσεις από την v2.1 (API Level 7) του android και μετά.

Οι διαφορετικές εκδόσεις τις ActionBar βρίσκονται σε διαφορετικά πακέτα με την κανονική έκδοση (API Level 11+) να βρίσκεται στο android.app.ActionBar και την support να βρίσκεται στο android.support.v7.app.ActionBar. Το μεγαλύτερο τμήμα των API τους είναι το ίδιο με μερικές διαφοροποιήσεις. Αυτό έχει γίνει εσκεμμένα ώστε όταν δεν θα υπάρχει ανάγκη υποστήριξης παλαιότερων συσκευών να μην χρειάζονται ιδιαίτερες τροποποιήσεις για τη μετάβαση στο κανονικό API.

2.2.1. ActionBarSherlock

Η άλλη διαθέσιμη επιλογή που ήταν δημοφιλής πριν την προσθήκη της ActionBar στα support libraries είναι η ActionBarSherlock.

Η ActionBarSherlock είναι μια αυτόνομη βιβλιοθήκη η οποία σχεδιάστηκε για να παρέχει τη δυνατότητα της χρήσης της action bar σε διαφορετικές εκδόσεις του Android με τη χρήση ενός ενοποιημένου API. Η βιβλιοθήκη είναι έργο του Jake Wharton και είναι διαθέσιμη κάτω από το Apache License Version 2.0. Υποστηρίζει όλες τις εκδόσεις του Android από την 2 και μετά. Η βιβλιοθήκη χρησιμοποιεί αυτόματα την ActionBar του συστήματος σε συσκευές που τρέχουν από την έκδοση 4.0 και μετά ενώ χρησιμοποιεί μια τροποποιημένη υλοποίηση της action bar η οποία βασίζεται στον πηγαίο κώδικα του Android 4 για να πετύχει την ίδια λειτουργικότητα σε παλαιότερες εκδόσεις.

Όταν δημιουργούμε ένα activity για να κάνουμε χρήση της action bar σε όλες τις εκδόσεις του android θα πρέπει η activity μας να κληρονομεί μια από τις Sherlock activity classes[6] (`SherlockActivity`, `SherlockFragmentActivity`). Η αλληλεπίδραση με την action bar γίνεται με τη χρήση της μεθόδου `getSupportActionBar()` αντί για την `getActionBar()`.

Το API που παρέχεται από την `ActionBar` (Sherlock) είναι ένα ακριβές αντίγραφο από το public API τις κανονικής `ActionBar`.

[6] Στην περίπτωση μας χρησιμοποιούνται τα RoboSherlock classes, τα οποία κληρονομούν τα Shelock classes και παράλληλα υλοποιούν τη λειτουργικότητα των Robo classes τα οποία είναι τμήμα του Roboguice.

Επιπλέον υπάρχουν fragment classes με το πρόθεμα Sherlock (π.χ. `SherlockFragment`, `SherlockListFragment`) οι οποίες πρέπει να χρησιμοποιηθούν για να υπάρχει σωστή λειτουργικότητα.

Για να παρέχει λειτουργικότητα η οποία δεν ήταν διαθέσιμη πριν από την έκδοση 3.0 η βιβλιοθήκη περιλαμβάνει και χρησιμοποιεί αρκετές classes οι οποίες διατηρούν τα ίδιο ονόματα με τις αντίστοιχες native μεθόδους. Οι περισσότερο συνηθισμένες είναι οι:

- `com.actionbarsherlock.app.ActionBar`
- `com.actionbarsherlock.view.Menu`
- `com.actionbarsherlock.view.MenuItem`
- `com.actionbarsherlock.view.MenuInflater`

2.2.2. Action Items

Η action bar παρέχει πρόσβαση στα πιο σημαντικές ενέργειες γύρω από το τρέχον πλαίσιο της εφαρμογής. Αυτά εμφανίζονται απευθείας στην action bar ως εικονίδια (με ή χωρίς κείμενο) και είναι γνωστά ως action buttons. Οι ενέργειες που δεν γίνεται να χωρέσουν στο μήκος της action bar η δεν είναι αρκετά σημαντικές κρύβονται κάτω από το κουμπί υπερχείλισης (overflow button) και ο χρήστης μπορεί να αποκτήσει πρόσβαση σε αυτές είτε με το πάτημα του κουμπιού, είτε πατώντας το κουμπί μενού της συσκευής (σε παλιότερες συσκευές όπου αυτό είναι διαθέσιμο.

Κατά την εκκίνηση του activity το σύστημα δημιουργεί τα action items καλώντας την μέθοδο `onCreateOptionsMenu()` του activity η οποία έχει τη δυνατότητα να δημιουργήσει τα action items χρησιμοποιώντας ένα xml menu resource στο οποίο ορίζονται τα action items.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <item android:id="@+id/actionbar_settings"
          android:title="@string/main_menu_settings"/>
    <item android:id="@+id/actionbar_help"
          android:title="@string/main_menu_help"/>
</menu>
```

Όπως βλέπουμε στο παράδειγμα ορίζονται items που αντιπροσωπεύουν 2 ενέργειες η μία είναι το άνοιγμα του μενού των ρυθμίσεων και η άλλη της βοήθειας.

```
public boolean onCreateOptionsMenu(Menu menu) {
  MenuInflater inflater = getSupportMenuInflater();
  inflater.inflate(R.menu.menu, menu);
  return true;
}
```

Όπως μπορούμε να δούμε η `onCreateOptionsMenu()` διαβάζει το menu layout και δημιουργεί τα αντικείμενα για κάθε item (κατά το inflate), και τα προσθέτει στην action bar.

Το item μας παρέχει τη δυνατότητα να δηλώσουμε αν ένα στοιχείο θα πρέπει να εμφανιστεί στην action bar αυτό μπορεί να γίνει θέτοντας την ιδιότητα (attribute) `android:showAsAction=` στο στοιχείο item. Αν για παράδειγμα θέσουμε `android:showAsAction="ifRoom"` τότε το στοιχείο θα εμφανιστεί στην action bar μόνο στην περίπτωση που υπάρχει διαθέσιμος χώρος.

Όταν ο χρήστης πατήσει πάνω σε κάποιο από τα εικονίδια της action bar τότε το σύστημα καλεί τη μέθοδο `onOptionsItemSelected()` του activity που είναι ενεργό. Χρησιμοποιώντας την τιμή της παραμέτρου `MenuItem` που περνά ως όρισμα στη μέθοδο μπορούμε να ξεχωρίσουμε την ενέργεια με την κλήση της μεθόδου `getItemId()` η οποία θα επιστρέψει το μοναδικό αναγνωριστικό που χαρακτηρίζει το item όπως αυτό έχει οριστεί στην ιδιότητα id.

```
@Override public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
   case android.R.id.home:
        if (mDrawerLayout.isDrawerOpen(mDrawerMenu)) {
              mDrawerLayout.closeDrawer(mDrawerMenu);
        } else {
              mDrawerLayout.openDrawer(mDrawerMenu);
        }

        return true;
   case R.id.actionbar_settings:
        startActivity(new Intent(this,  AppPreferenceView.class));
        return true;
   case R.id.actionbar_help:
        Intent openHelp = new Intent(Intent.ACTION_VIEW);
        openHelp.setData(Uri.parse("http://kelsos.net/musicbeeremote/help/"));
        startActivity(openHelp);
        return true;
   default:
        return false;
   }
}
```

Όπως μπορούμε να δούμε, με τη χρήση του id μπορούμε να αναγνωρίσουμε την ενέργεια του χρήστη (το πλήκτρο που πατήθηκε) και να εκτελέσουμε το αντίστοιχο τμήμα κώδικα. Στην περίπτωση του `android.R.id.home` το οποίο αντιπροσωπεύεται από το εικονίδιο της εφαρμογής υπάρχει ο κώδικας που ανοίγει ή κλείνει το navigation drawer, το `R.id.actionbar_settings` αντιπροσωπεύει την επιλογή settings που βρίσκεται στο overflow menu ξεκινά το settings activity ενώ το `R.id.actionbar_help` ανοίγει τη σελίδα βοήθειας στον browser της συσκευής.

2.2.3. Home Icon

Το εικονίδιο της εφαρμογής μπορεί να χρησιμοποιηθεί για την περιήγηση στην ιεραρχία των διαφορετικών οθονών της εφαρμογής, αυτό επιτυγχάνεται καλώντας τη μέθοδο `getSupportActionBar().setDisplayHomeAsUpEnabled(true)`. Επειδή η περιήγηση στο βασικό τμήμα τις εφαρμογής έχει υλοποιηθεί με τη χρήστη του navigation drawer, αυτή συμπεριφορά είναι ορατή μόνο όταν ο χρήστης ανοίξει το παράθυρο των ρυθμίσεων. Πατώντας το εικονίδιο ο χρήστης θα πρέπει να επιστρέψει σε μια προηγούμενη κατάσταση.

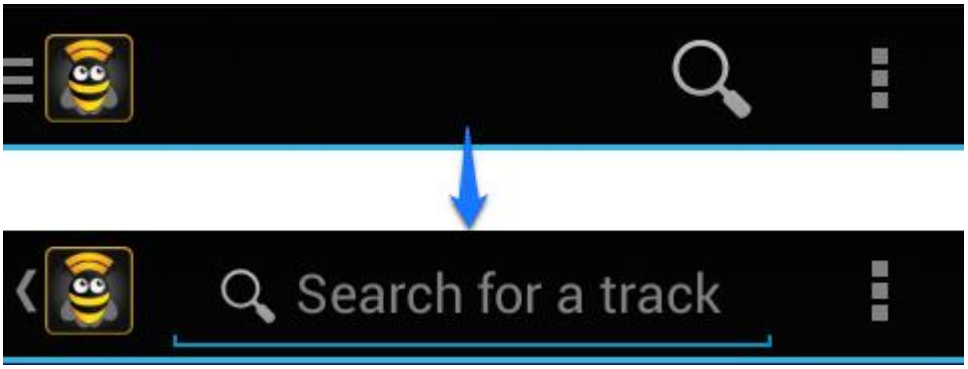
Στην περίπτωση μας το πάτημα του κουμπιού καλεί την μέθοδο `finish()` του activity που σηματοδοτεί ότι το activity έχει ολοκληρώσει τη δραστηριότητα του και πρέπει να κλείσει. Αυτό έχει ως αποτέλεσμα την επιστροφή στην προηγούμενη χρονολογικά κατάσταση (όπως θα γινόταν με το πάτημα του πλήκτρου back).

Στο γενικό κανόνα η συμπεριφορά του up πλήκτρου είναι διαφορετική ως προς το back πλήκτρο υπό την έννοια ότι το back αντιπροσωπεύει την τη χρονολογική σχέση των διαφορετικών views (σειρά με την οποία εμφανίστηκαν) ενώ το πλήκτρο up αντιπροσωπεύει την ιεραρχική δομή της εφαρμογής.

Εικόνα 4. Home as up

2.2.4. Action Views

Ένα action view είναι ένα widget το οποίο εμφανίζεται στην action bar ως υποκατάστατο για ένα κουμπί. Τα action views προσφέρουν γρήγορη πρόσβαση σε ενέργειες χωρίς να απαιτείται η αλλαγή fragment ή activity, και χωρίς να αντικατασταθεί η action bar. Αν για παράδειγμα θέλουμε να χρησιμοποιήσουμε αναζήτηση μπορούμε να προσθέσουμε ένα action view το οποίο περιλαμβάνει ένα `SearchView`.



Εικόνα 5. Action bar με SearchView

Η υλοποίηση ενός action view μπορεί να διαφέρει ανάλογα με το ποια υλοποίηση (ActionBar, ActionBar Support ή ActionBarSherlock) έχει χρησιμοποιηθεί.

Στην περίπτωση μας χρησιμοποιώντας τη βιβλιοθήκη ActionBarSherlock δηλώνουμε αρχικά το menu item το οποίο αντιπροσωπεύει στο action view μας.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:icon="@drawable/abs__ic_search"
            android:id="@+id/now_playing_search_item"
            android:showAsAction="collapseActionView|always"
            android:title="@string/now_playing_search"/>
</menu>
```

Όπως μπορούμε να δούμε στο showAsAction attribute δηλώνουμε ότι το item είναι collapse action view και ότι αυτό θα πρέπει να είναι πάντα (always) εμφανές στην action bar.

```
@Override public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {

  mSearchView = new SearchView(((RoboSherlockFragmentActivity)
getActivity()).getSupportActionBar().getThemedContext());
  mSearchView.setQueryHint(getString(R.string.now_playing_search_hint));
  mSearchView.setIconifiedByDefault(true);

  inflater.inflate(R.menu.menu_now_playing, menu);
  mSearchItem = menu.findItem(R.id.now_playing_search_item);
  mSearchItem.setActionView(mSearchView);
  mSearchView.setOnQueryTextListener(this);
}
```

Ο constructor της SearchView απαιτεί ως παράμετρο το context, επειδή όμως πρόκειται να εμφανιστεί στην action bar του δίνουμε το αποτέλεσμα της getThemedContext() της action bar. Το themed context έχει το κατάλληλο θέμα για όλα τα αντικείμενα (views) που πρόκειται να εμφανιστούν στην action bar και φροντίζει ότι έχουν την κατάλληλη αντίθεση ώστε να εμφανιστούν εκεί.

Με τη μέθοδο setQueryHint() δηλώνουμε το μήνυμα που θα εμφανίζεται στο χρήστη αφού εμφανιστεί το πεδίο αναζήτησης και πριν πληκτρολογήσει κάτι (μια μικρή περιγραφή του τι ακριβώς κάνει το συγκεκριμένο πεδίο). Η μέθοδος setIconifiedByDefault() ορίζει ότι αρχικά ο action view θα βρίσκεται σε κατάσταση εικονιδίου (δηλαδή το πεδίο αναζήτησης δεν θα είναι εμφανές).

Στη συνέχεια αφού κάνουμε inflate το menu και προσθέσουμε το αντικείμενο στην action bar βρίσκουμε το αντικείμενο και ορίζουμε ότι το action view του αντικειμένου είναι το SearchView. Στην συνέχεια ορίζουμε τον listener για τα user actions του SearchView στο αντικείμενο της class NowPlayingFragment.

Πηγαίνοντας στην class NowPlayingFragment μπορούμε να δούμε ότι αυτή υλοποιεί λειτουργικότητα του interface SearchView.OnQueryTextListener. Το interface αυτό διαθέτει δύο μεθόδους που είναι απαραίτητο να υλοποιήσουμε. Η πρώτη είναι η onQueryTextSubmit() η οποία δέχεται ένα string (το κείμενο που έχουμε γράψει στο πεδίο εισόδου) και εκτελείται όταν ο χρήστης πατήσει το πλήκτρο τις αναζήτησης.

```
public boolean onQueryTextSubmit(String query) {
  bus.post(new MessageEvent(ProtocolEventType.UserAction, new
UserAction(Protocol.NowPlayingListSearch, query.trim())));
  mSearchView.setIconified(true);
  mSearchItem.collapseActionView();
  return false;
}
```

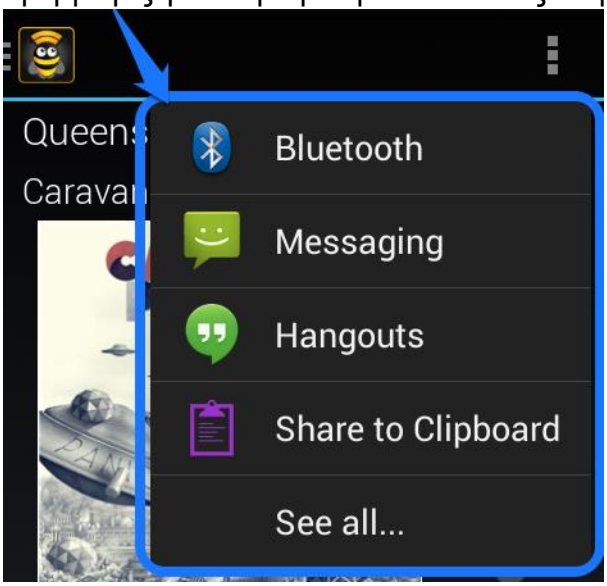
Κατά την εκτέλεση της μεθόδου στέλνουμε το κατάλληλο μήνυμα στο event bus. Στη συνέχεια επαναφέρουμε το action view στην αρχική του κατάσταση ως εικονίδιο με τη χρήση των μεθόδων `setIconified()` και `collapseActionView()`.

Η δεύτερη μέθοδος είναι η `onQueryTextChange()` η οποία εκτελείται καθώς αλλάξει το κείμενο αναζήτηση και παίρνει ως παράμετρο το νέο κείμενο, η χρήση της είναι για την περίπτωση που έχουμε για παράδειγμα τοπικά δεδομένα και θέλουμε να τα φιλτράρουμε δυναμικά καθώς ο χρήστης πληκτρολογεί.

2.2.5. Action Providers

Παρόμοια με τα action views ένας action provider αντικαθιστά ένα κουμπί με ένα τροποποιημένο layout. Σε αντίθεση με ένα action view ο action provider ελέγχει όλες τις συμπεριφορές της δραστηριότητα και μπορεί να παρουσιάσει ένα sub menu όταν πατηθεί. Για να δηλωθεί ένας action provider αρκεί να δοθεί το πλήρες όνομα της κλάσης (συμπεριλαμβανομένου και του πακέτου στο οποίο βρίσκεται) στην ιδιότητα `actionViewClass` του αντικειμένου (`item`) που αντιπροσωπεύει το μενού.

Έχουμε τη δυνατότητα να δημιουργήσουμε τους δικούς μας action providers επεκτείνοντας τη λειτουργικότητα της κλάσης `ActionProvider`, παρόλα αυτά το Android παρέχει κάποιους έτοιμους action providers όπως για παράδειγμα ο `ShareActionProvider`, ο οποίος διευκολύνει το διαμοιρασμό (share) δείχνοντας μια λίστα με τις διαθέσιμες εφαρμογές για διαμοιρασμό απευθείας στην action bar.



Εικόνα 6. ShareActionProvider

Επειδή η κάθε `ActionProvider` κλάση ορίζει τις δικές της συμπεριφορές, δεν χρειάζεται να ακούμε για την ενέργεια μέσα στην `onOptionsItemSelected()`. Αν είναι απαραίτητο μπορούμε να ακούσουμε για click events μέσα στην μέθοδο (για παράδειγμα στην περίπτωση που χρειάζεται να εκτελέσουμε κάποια άλλη δραστηριότητα παράλληλα), απλά θα πρέπει η μέθοδος να επιστρέψει false ώστε ο action provider να λάβει το `onPerformDefaultAction()` callback και να εκτελέσει την προκαθορισμένη του δραστηριότητα.

2.2.6. ShareActionProvider

Η προσθήκη ενός `ShareActionProvider` σε ένα project μπορεί να έχει μικρές διαφοροποιήσεις ανάλογα με το αν γίνεται είτε με κάποια από τις δυο βιβλιοθήκες (ABS, Android Support) ή με τον native τρόπο εφόσον στοχεύουμε σε συσκευές που τρέχουν τουλάχιστον την έκδοση 3.0 του Android.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
       android:id="@+id/actionbar_share"
       android:title="@string/main_menu_title_share"
  android:actionProviderClass="com.actionbarsherlock.widget.ShareActionProvider
"/>
</menu>
```

Για να προσθέσουμε τη δυνατότητα διαμοιρασμού με τη χρήση του ShareActionProvider αρχικά θα πρέπει να δηλώσουμε στην ιδιότητα actionProviderClass του <item> που αντιπροσωπεύει το κουμπί διαμοιρασμού, το πλήρες όνομα της κλάσης του action provider. Έτσι ο provider αποκτά τον πλήρη έλεγχο του αντικειμένου και διαχειρίζεται πλήρως την εμφάνιση και την συμπεριφορά του. Πρέπει μόνο να δοθεί ο τίτλος του στοιχείου, ο οποίος θα εμφανιστεί στην περίπτωση που η δραστηριότητα εμφανιστεί στο menu (action overflow).

```
@Override public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
   case R.id.actionbar_share:
       Intent shareIntent = new Intent(Intent.ACTION_SEND);
       shareIntent.setType("text/plain");
       shareIntent.putExtra(Intent.EXTRA_TEXT, "Now Playing: " +
artistLabel.getText() + " - " + titleLabel.getText());
       setShareIntent(shareIntent);
       return true;
   default:
       return false;
  }
}
```

Όταν ο χρήστης πατήσει το πλήκτρο για το διαμοιρασμό (share) δημιουργούμε ένα καινούργιο Intent με Intent.ACTION_SEND θέτουμε τον τύπο του ως “text/plain” και στα extra του intent περνάμε τα δεδομένα που θέλουμε να μοιραστούμε σε κάποια από την εφαρμογές.

```
private void setShareIntent(Intent shareIntent) {
  if (mShareActionProvider != null)
mShareActionProvider.setShareIntent(shareIntent);
}
```

Στη συνέχεια θέτουμε το shareIntent στο instance του ShareActionProvider με τη μέθοδο setShareIntent().

2.3 Navigation Drawer

Το navigation drawer είναι ένα πλαίσιο το οποίο εμφανίζεται από την αριστερή πλευρά τις οθόνης και εμφανίζει τα βασικά στοιχεία περιήγησης στην εφαρμογή.

Ο χρήστης μπορεί να εμφανίσει το navigation drawer περνώντας το δάκτυλο του από την αριστερή πλευρά τις οθόνης προς τα δεξιά, ή πατώντας το εικονίδιο της εφαρμογής στην ActionBar.

Για να προσθέσουμε ένα navigation drawer θα πρέπει να δηλώσουμε ένα `DrawerLayout` σαν το κομβικό αντικείμενο στο layout μας και εσωτερικά να δηλώσουμε ένα στοιχείο το οποίο περιέχει το βασικό περιεχόμενο της εφαρμογής μας, καθώς και ένα ακόμη το οποίο περιέχει το navigation drawer.

```
<?xml version="1.0" encoding="utf-8"?>

<android.support.v4.widget.DrawerLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/drawer_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    <FrameLayout
            android:id="@+id/fragment_container"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1"
            />

    <fragment
            android:id="@+id/drawer_menu"
            android:layout_width="240dp"
            android:layout_height="match_parent"
            android:layout_gravity="start"
            android:name="com.kelsos.mbrc.ui.fragments.DrawerFragment"
            tools:layout="@layout/ui_fragment_drawer"/>


</android.support.v4.widget.DrawerLayout>
```

Όπως μπορούμε να δούμε υπάρχει το `FrameLayout` `fragment_container` το οποίο στο οποίο προσθέτουμε δυναμικά τα fragments μας κατά το χρόνο εκτέλεσης, καθώς και ένα ακόμη fragment το `DrawerFragment` το οποίο περιέχει τη λειτουργικότητα του navigation drawer.

Στο layout μπορούμε να δούμε μερικά βασικά χαρακτηριστικά για ένα layout. Αρχικά το τμήμα που περιέχει το βασικό περιεχόμενο (`FrameLayout`) θα πρέπει να είναι το πρώτο παιδί του `DrawerLayout` λόγο της σειράς . Το βασικό περιεχόμενο θα πρέπει να ταιριάζει σε ύψος και πλάτος με το γονέα του μιας και αντιπροσωπεύει όλο το user interface όταν το navigation drawer είναι κρυμμένο. Το drawer ορίζει το πλάτος του σε dp (Density-independent pixels) και το ύψος του ταιριάζει με το ύψος του view γονέα, παρόλα αυτά το πλάτος καλό θα ήταν να μην ξεπερνά τα 320 dp ώστε ένα τμήμα του βασικού περιεχομένου να είναι πάντοτε ορατό στο χρήστη.

```
mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
mDrawerMenu = findViewById(R.id.drawer_menu);

mDrawerToggle = new ActionBarDrawerToggle(this, mDrawerLayout,
   R.drawable.ic_drawer, R.string.drawer_open, R.string.drawer_close) {
  public void onDrawerClosed(View view) {
   invalidateOptionsMenu();
   if (navChanged) {
       navigateToView();
   }
  }

  public void onDrawerOpened(View view) {
   invalidateOptionsMenu();
  }
};

mDrawerLayout.setDrawerListener(mDrawerToggle);
mDrawerLayout.setDrawerShadow(R.drawable.drawer_shadow, 1);
```

Μπορούμε να ακούσουμε για τα γεγονότα που αντιπροσωπεύουν το άνοιγμα και το κλείσιμο του drawer καλώντας την μέθοδο `setDrawerListener()` και περνώντας της ως παράμετρο μια υλοποίηση του interface `DrawerLayout.DrawerListener`. Το interface αυτό παρέχει τα κατάλληλα callbacks για τα γεγονότα του drawer. Παρόλα αυτά αντί να υλοποιήσουμε το interface στην εφόσον χρησιμοποιούμε την `ActionBar` μπορούμε απλά να επεκτείνουμε την κλάση `ActionBarDrawerToggle`. Η συγκεκριμένη κλάση εκτός από το ότι μας παρέχει τη δυνατότητα να υλοποιήσουμε τις callback μεθόδους για το άνοιγμα και το κλείσιμο του drawer διαθέτει και την κατάλληλη διάδραση μεταξύ του εικονιδίου της εφαρμογής και του drawer.

Κατά το navigation δηλαδή όταν χρήστης πατήσει κάποια επιλογή στο drawer αντικαθιστάτε δυναμικά το περιεχόμενο του `frame_container`, αν η αλλαγή του fragment ξεκινήσει προτού κλείσει το drawer το αποτέλεσμα είναι ένα οπτικό κόλλημα. Η αλλαγή πρέπει να ξεκινήσει αφού κλείσει το drawer. Για το λόγο αυτό όταν ο χρήστης διαλέξει μια νέα επιλογή (αν πατήσει την ήδη επιλεγμένη το μόνο που θα συμβεί είναι να κλείσει το drawer), το `DrawerFragment` το οποίο περιέχει τις επιλογές για το navigation μέσα στην εφαρμογή θα ειδοποιήσει το `MainFragmentActivity` με ένα event (`DrawerEvent`).

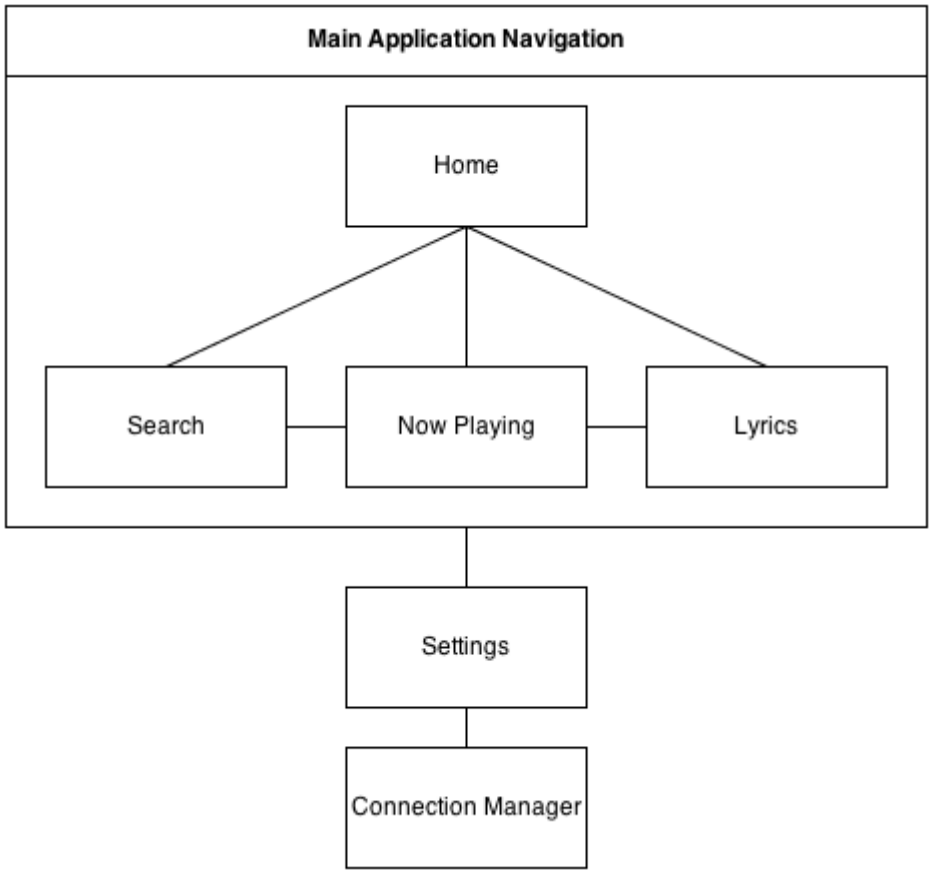
```
@Subscribe public void handleDrawerEvent(DrawerEvent event) {
  if (event.isCloseDrawer()) {
   closeDrawer();
  } else {
   navChanged = true;
   mDisplay = event.getNavigate();
   closeDrawer();
  }
}
```

Η μέθοδος που χειρίζεται το event είτε θα κλείσει το drawer αν η μέθοδος `isCloseDrawer()` γυρίσει true είτε θα αλλάξει την τιμή της μεταβλητής `navChanged` σε true και θα αποθηκεύσει το πού πρέπει να πάει στη μεταβλητή `mDisplay`, έπειτα θα κλείσει το drawer. Και έτσι επιστρέφουμε στη μέθοδο `onDrawerClosed()` της κλάσης `ActionBarDrawerToggle` που ελέγχει τη τιμή της μεταβλητής `navChanged`

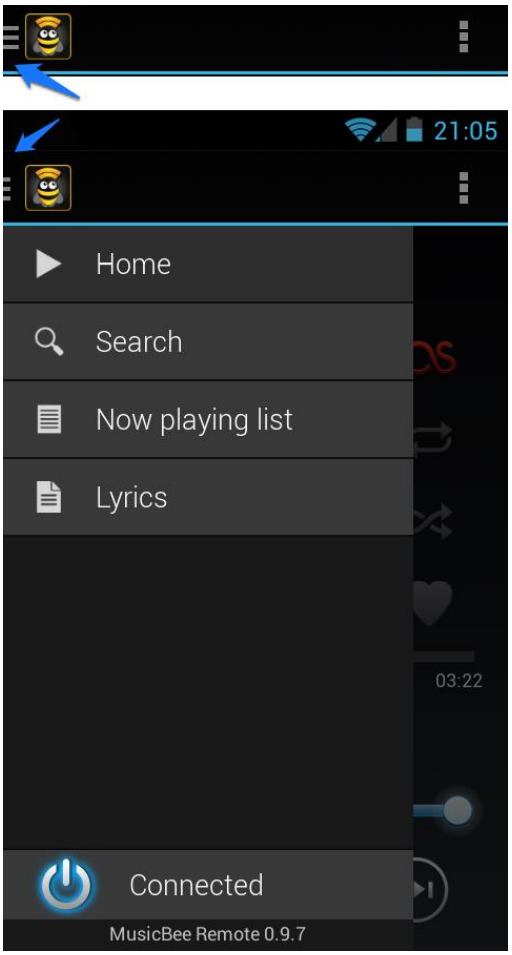
και αν είναι true τότε καλεί την μέθοδο `navigateToView()`, η οποία με τη σειρά της είτε αντικαθιστά δυναμικά το fragment είτε καλεί τη μέθοδο `onBackPressed()` για να πάει στην προηγούμενη εγγραφή του back stack αν πατηθεί το πλήκτρο home.

Στο Εικόνα 7 μπορούμε να δούμε το πώς είναι η ιεραρχία της περιήγησης μέσα στην εφαρμογή. Στο ανώτερο επίπεδο έχουμε το home το οποίο αντιπροσωπεύει τη βασική οθόνη της εφαρμογής. Στη συνέχει έχουμε στο επόμενο επίπεδο τα search, now playing και lyrics τα οποία έχουν επιπλέον πληροφορίες. Με τη χρήση του navigation drawer ο χρήστης μπορεί να μεταβεί σε οποιοδήποτε από αυτά τα views. Για παράδειγμα αν βρίσκεται στο home μπορεί να μεταβεί στο search αλλά αν έπειτα πατήσει το πλήκτρο back τότε θα επιστρέψει στο home. Τώρα αν από το search μεταβεί για παράδειγμα στο now playing και πατήσει το back τότε δεν θα επιστρέψει στο search αλλά στο home το οποίο βρίσκεται ιεραρχικά πάνω του.

Οι ρυθμίσεις λόγο σύμβασης βρίσκονται στην ActionBar και ως εκ τούτου είναι διαθέσιμες από όλα τα fragments (views) του βασικού navigation.



Εικόνα 7. Η ιεραρχία των views στην εφαρμογή



Εικόνα 8. Navigation Drawer και η αλλαγή του application icon

2.4 Notifications

Ένα notification είναι ένα μήνυμα που μπορεί να παρουσιαστεί στο χρήστη της εφαρμογής έξω από το κανονικό περιβάλλον της εφαρμογής. Όταν ζητάμε από το σύστημα να εμφανίσει ένα notification αρχικά το εμφανίζει ως εικονίδιο στην περιοχή ειδοποιήσεων. Για να δει τις λεπτομέρειες ενός notification ο χρήστης χρειάζεται να ανοίξει το notification drawer. Οι δύο αναφερθείς περιοχές είναι ελεγχόμενες από το σύστημα και ο χρήστης έχει πρόσβαση σε αυτές ανά πάσα στιγμή.

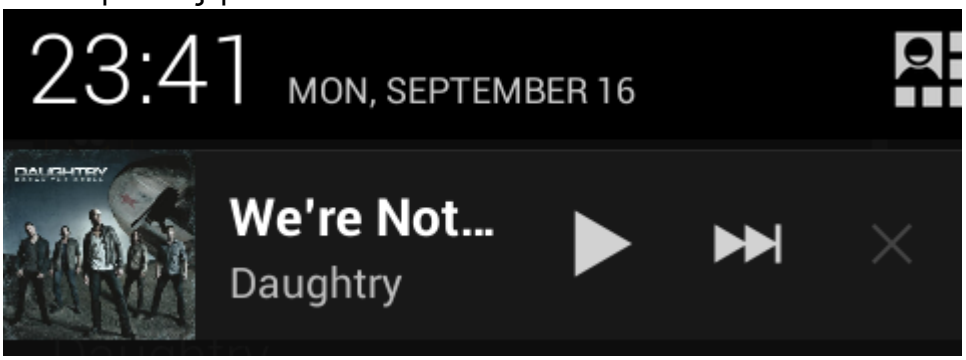
Εικόνα 9. Περιοχή ειδοποιήσεων (notification area)

Τα notifications στο drawer μπορούν να εμφανιστούν σε δύο διαφορετικά οπτικά στυλ ανάλογα με την έκδοση και την κατάσταση του drawer. Υπάρχει το κανονικό view (normal view) το οποίο είναι το standard view των notification στο notification drawer και το μεγάλο view (big view) το οποίο εμφανίζεται όταν το

notification έχει επεκταθεί. Τα μεγάλα notifications έχουν προστεθεί από την έκδοση 4.1 του Android (API Level 16) και μετά.

2.4.1 Normal View

Ένα normal view notification εμφανίζεται σε μια περιοχή η οποία έχει μέγιστο ύψος 64 dp. Ακόμα και αν δημιουργηθεί ένα notification χρησιμοποιώντας το big view αυτό θα παραμείνει στο normal μέχρι να επεκταθεί. Στο Εικόνα 10 μπορούμε να δούμε πως φαίνεται το normal view.

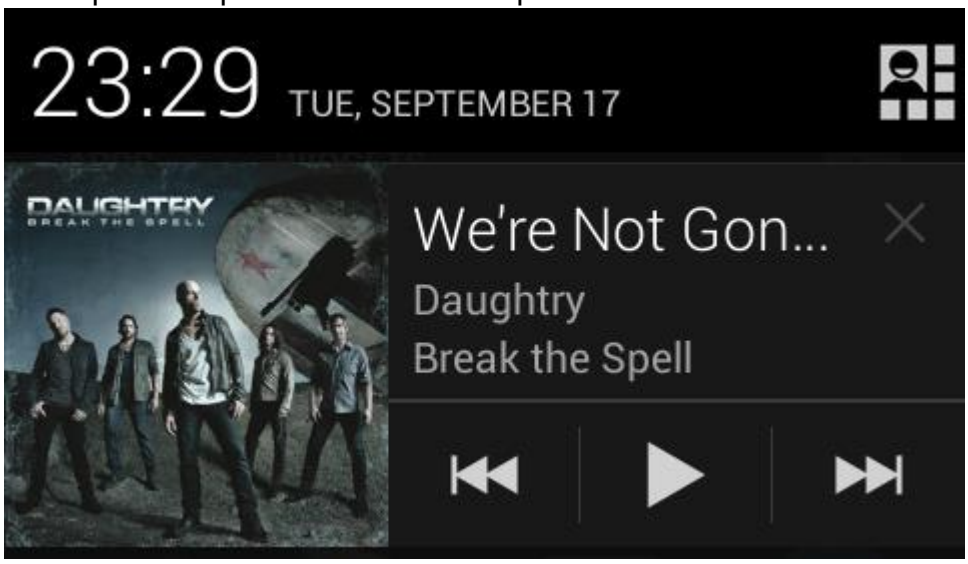


Εικόνα 10. Normal view notification

Το notification χρησιμοποιεί ένα custom xml layout (`ui_notification_control.xml`) για να παρουσιάσει το περιεχόμενο.

2.4.2 Big View

Το big view ενός notification εμφανίζεται μόνο όταν αυτό επεκτείνετε, το οποίο μπορεί να συμβεί είτε όταν ένα notification βρίσκεται στην κορυφή του drawer, είτε όταν επεκταθεί από το χρήστη. Αυτού του είδους τα notification είναι διαθέσιμα από την έκδοση 4.1 του Android και μετά.



Εικόνα 11. Big view notification

2.4.3 Δημιουργία notification

Μπορούν να οριστούν πληροφορίες για το γραφικό τμήμα καθώς και συμπεριφορές για ένα notification στο αντικείμενο `NotificationCompat.Builder` (λόγω του ότι η class `Notification.Builder` προστέθηκε στην έκδοση 3.0 του Android για υποστήριξη σε συσκευές που τρέχουν παλαιότερες εκδόσεις υπάρχει

η `NotificationCompat.Builder` που είναι τμήμα της Android Support Library v4). Για να δημιουργηθεί το notification πρέπει να κληθεί η μέθοδος `NotificationCompat.Builder.build()`, η οποία επιστρέφει ένα `Notification` αντικείμενο, το οποίο δημιουργείται με βάση τις παραμέτρους που έχουμε δώσει στο `NotificationCompat.Builder`. Για να εμφανιστεί το notification πρέπει να περάσουμε το αντικείμενο στο σύστημα καλώντας τη μέθοδο `NotificationManager.notify()`.

Ένα `Notification` αντικείμενο θα πρέπει απαραίτητα να συμπεριλαμβάνει τα ακόλουθα στοιχεία. Ένα μικρό εικονίδιο το οποίο ορίζεται με τη χρήση της μεθόδου `setSmallIcon()`, ένα τίτλο ο οποίος ορίζεται με τι χρήση της μεθόδου `setContentTitle()` και τέλος μια λεπτομερή περιγραφή η οποία ορίζεται με τη μέθοδο `setContentText()`.

2.4.4 Ενέργειες

Αν και είναι προαιρετικό, υπάρχει η δυνατότητα να προστεθεί τουλάχιστον μια ενέργεια σε ένα notification. Αυτό μπορεί να δώσει τη δυνατότητα στους χρήστες για παράδειγμα να ανοίξουν κάποιο activity της εφαρμογής απευθείας μέσα από το notification.

Ένα notification μπορεί να παρέχει πολλαπλές ενέργειες (καλό θα ήταν να σημειωθεί ότι αυτό το χαρακτηριστικό δεν είναι διαθέσιμο σε συσκευές που τρέχουν την έκδοση 2.x του Android). Καλό θα ήταν να ορίζεται πάντα η ενέργεια που εκτελείται όταν ο χρήστης πατήσει πάνω στο notification, αυτό θα μπορούσε για παράδειγμα να τον πηγαίνει στην κεντρική οθόνη της εφαρμογής.

Από την έκδοση 4.1 και μετά υπάρχει η δυνατότητα για την προσθήκη κουμπιών τα οποία εκτελούν επιπλέον ενέργειες όπως για παράδειγμα την άμεση απάντηση σε κάποιο μήνυμα. Αν οριστούν επιπλέον κουμπιά θα πρέπει να οριστεί και η λειτουργικότητα η οποία εκτελείται κατά το πάτημα τους.

Μέσα σε ένα Notification η ενέργεια ορίζεται από ένα `PendingIntent` το οποίο περιέχει ένα Intent το οποίο με τη σειρά του ξεκινά ένα `Activity` της εφαρμογής. Για να συνδέσουμε ένα `PendingIntent` με μια ενέργεια πρέπει να καλέσουμε την κατάλληλή μέθοδο του `NotificationCompat.Builder`. Για παράδειγμα αν θέλουμε να ξεκινήσουμε ένα `Activity` όταν ο χρήστης πατήσει το κείμενο του notification στο drawer, πρέπει να καλέσουμε την `setContentIntent()` και να περάσουμε ως παράμετρο το `PendingIntent` που μας ενδιαφέρει.

2.4.5 Διαχείριση και ενημέρωση

Όταν χρειάζεται να εμφανιστεί ένα notification αρκετές φορές για τον ίδιο τύπο γεγονότος καλό θα ήταν να αποφευχθεί η επαναδημιουργία του. Θα ήταν καλύτερο να ενημερωθεί το προηγούμενο, είτε αλλάζοντας κάποιες από τις τιμές του, είτε προσθέτοντας σε αυτό, ή και τα δύο.

Ένα notification για να έχει τη δυνατότητα να ενημερωθεί θα πρέπει να εμφανιστεί με κάποιο ID. Για να γίνει αυτό πρέπει να κληθεί η μέθοδος `NotificationManager.notify(ID, notification)`. Για να ενημερώσουμε ένα notification από τη στιγμή που θα το εμφανίσουμε χρειάζεται είτε να ενημερώσουμε είτε να δημιουργήσουμε ένα νέο `NotificationCompat.Builder` αντικείμενο και να το εκδώσουμε χρησιμοποιώντας ξανά το ίδιο ID. Στην περίπτωση που το notification είναι ακόμη εμφανές το σύστημα θα το ενημερώσει

με τα περιεχόμενα του αντικειμένου, σε αντίθετη περίπτωση θα δημιουργήσει ένα νέο.

2.4.6 Custom Notifications

Το framework των notification παρέχει τη δυνατότητα για τη δημιουργία custom notification layout (xml layout), τα οποία ορίζουν την εμφάνισή ενός notification σε ένα `RemoteViews` αντικείμενο.

Το ύψος που είναι διαθέσιμο για τα custom notification layout εξαρτάται από το μέγεθος του notification view και είναι 64 dp για normal view και 256 για expanded.

Για να οριστεί ένα custom notification layout χρειάζεται να δημιουργηθεί αρχικά ένα `RemoteViews` αντικείμενο το οποίο κάνει inflate το xml layout. Αν χρησιμοποιούμε μια πρόσφατη έκδοση του Android (4.1 ή νεότερη) και χρησιμοποιούμε κουμπιά στο custom layout μπορούμε να καλέσουμε την μέθοδο `setOnClickPendingIntent()` στο `RemoteViews` αντικείμενο μας ώστε να ορίσουμε `PendingIntent` για το πάτημα του κουμπιού.

2.5 ViewPager

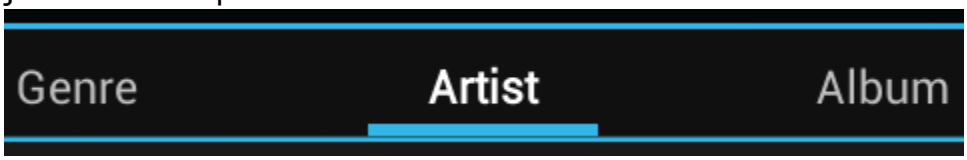
Ένα `ViewPager` χρησιμοποιείται για τη δημιουργία slides μεταξύ διαφορετικών views. Για παράδειγμα έχουμε ένα παρόμοια `ListFragments` τα οποία περιέχουν παρόμοια δεδομένα και θέλουμε να μπορούμε με κάποιο swipe gesture να μετακινηθούμε από το ένα fragment στο άλλο.

Το `ViewPager` έχει ήδη ενσωματωμένα τα απαιτούμενα gestures για την εναλλαγή σελίδων, και χρησιμοποιεί animation κατά την αλλαγή οπότε δεν απαιτείται από το χρήστη να δημιουργήσει κάτι. Για να λειτουργήσει το `ViewPager` απαιτείται η χρήση ενός `PagerAdapter` ο οποίος παρέχει τις σελίδες που παρουσιάζονται στο `ViewPager`.

Η δημιουργία ενός ViewPager απαιτεί αρχικά την δημιουργία ενός xml layout:

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:tools="http://schemas.android.com/tools"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical">

    <com.viewpagerindicator.TitlePageIndicator
            android:id="@+id/search_categories"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"
            android:background="#101010"/>

    <android.support.v4.view.ViewPager
            android:id="@+id/search_pager"
            android:layout_width="match_parent"
            android:layout_height="0dip"
            android:layout_weight="1"/>

        ...
</LinearLayout>
```

Προτού συνεχίσουμε καλό θα ήταν να γίνει μια αναφορά στην βιβλιοθήκη `ViewPageIndicator` η οποία παρέχει μια πολύ απλή λειτουργία όπως φαίνεται και στο Εικόνα 12. ViewPagerIndicator. Η λειτουργία του είναι να δείχνει τους τίτλους των σελίδων με μια επισήμανση στην τρέχουσα σελίδα. Πατώντας στον τίτλο μιας άλλης σελίδας πέρα από την τρέχουσα ο χρήστης μεταφέρεται στη σελίδα της οποίας τον τίτλο πάτησε.



Εικόνα 12. ViewPagerIndicator

Συνεχίζοντας όσον αφορά το ViewPager έχουμε την `SearchPagerAdapter` η οποία επεκτείνει την abstract class `FragmentStatePagerAdapter`.

Ο `FragmentStatePagerAdapter` διαχειρίζεται την αποθήκευση και την επαναφορά του state ενός fragment και είναι περισσότερο χρήσιμη όταν υπάρχει διαθέσιμος μεγάλος αριθμός σελίδων. Όταν οι σελίδες δεν είναι ορατές στο χρήστη, το fragment μπορεί να καταστραφεί διατηρώντας μόνο το state του. Αυτό έχει ως αποτέλεσμα να χρησιμοποιείται λιγότερη μνήμη για κάθε σελίδα συγκριτικά με τον `FragmentPagerAdapter`, προσθέτοντας πιθανότητα κάποια καθυστέρηση κατά την εναλλαγή σελίδων.

Ο adapter μας δίνει την καινούργια σελίδα κάθε φορά που αυτή αλλάζει. Αυτό συμβαίνει μέσω της μεθόδου `getItem()` η οποία παρέχει instances των fragments που εμφανίζονται ως σελίδες του `ViewPager`. Επίσης κατά την υλοποίηση του adapter είναι απαραίτητο να υλοποιηθεί και η μέθοδος `getCount()` η οποία επιστρέφει τον αριθμό των σελίδων που δημιουργούνται από τον adapter.

Στη συνέχει μπορούμε να μεταφερθούμε στο `SearchFragment`. Στην `onCreate()` δημιουργούμε ένα νέο instance του adapter. Έπειτα μέσα στην `onCreateView()` του fragment κάνουμε inflate το layout (αν χρησιμοποιήσουμε το `ViewPager` απευθείας σε κάποιο activity τότε το inflate γίνεται με την κλήση της μεθόδου `setContentView()` που καλείται στην `onCreate()` του activity).

```
@Override public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
  View view = inflater.inflate(R.layout.ui_fragment_search, container, false);
  mPager = (ViewPager) view.findViewById(R.id.search_pager);
  mPager.setAdapter(mAdapter);
  TitlePageIndicator titleIndicator = (TitlePageIndicator)
view.findViewById(R.id.search_categories);
  titleIndicator.setViewPager(mPager);
  return view;
}
```

Ορίζουμε τον adapter στο `ViewPager` με τη χρήση της μεθόδου `setAdapter()` και στην συνέχει ορίζουμε τον pager στον `TitlePageIndicator` με τη χρήση της μεθόδου `setViewPager()` ώστε να πετύχουμε τη λειτουργία του `ViewPagerIndicator`.

Επίλογος

Στο κεφαλαίο αυτό είδαμε τμήματα του Android SDK τα οποία αποτελούν σε συνδυασμό με το Holo Theme τα βασικά στοιχεία από τα οποία αποτελείται μια σύγχρονη εφαρμογή για την πλατφόρμα Android. Έγινε μια παρουσίαση των υλοποιήσεων όπως αυτές γίνονται με τη χρήση των support libraries, που έχουν σχεδιαστεί ώστε να δίνεται η δυνατότητα στους προγραμματιστές να σχεδιάζουν τις εφαρμογές τους στοχεύοντας στις νεότερες εκδόσεις της πλατφόρμας, κάνοντας συγχρόνως χρήση όλων των νεότερων στοιχείων που προστίθενται στο SDK, χωρίς όμως να χάνουν την υποστήριξη συσκευών που τρέχουν παλαιότερες εκδόσεις του λειτουργικού.

Η μη ύπαρξη των support libraries θα δημιουργούσε δυνατό πρόβλημα μιας και όλα τα στοιχεία που παρουσιάστηκαν δεν θα ήταν δυνατό να χρησιμοποιηθούν αν θέλαμε να υποστηρίξουμε συσκευές που τρέχουν την έκδοση 2^7 του Android.

Στο επόμενο κεφάλαιο παρουσιάζονται οι βιβλιοθήκες που χρησιμοποιήθηκαν κατά την ανάπτυξη της εφαρμογής καθώς και κάποιες από τις δυνατότητες που παρέχουν, όπως για παράδειγμα το design pattern Dependency Injection.

[7] Αξίζει να σημειωθεί ότι το καλοκαίρι – φθινόπωρο του 2013 η έκδοση 2.3.x του Android τρέχει στο 30% των συσκευών σύμφωνα με επίσημα στατιστικά, ποσοστό καθόλου αμελητέο.

ΚΕΦΑΛΑΙΟ 3

ΒΙΒΛΙΟΘΗΚΕΣ

Εισαγωγή

Στο κεφάλαιο αυτό γίνεται μια παρουσίαση των βιβλιοθηκών που χρησιμοποιήθηκαν κατά την ανάπτυξη της εφαρμογής με εξαίρεση των ActionBarSherlock και Android Support Library v4 και του ViewPagerIndicator των οποίων χαρακτηριστικά παρουσιάστηκαν στο δεύτερο κεφάλαιο. Μαζί με τις βιβλιοθήκες γίνετε μια σύντομη παρουσίαση του dependency injection pattern στα πλαίσια της χρήσης της βιβλιοθήκης Roboguice.

Στη συνέχεια γίνεται μια σύντομη παρουσίαση της χρήσης του Otto Event bus το οποίο είχε σημαντικό ρόλο στην εσωτερική λειτουργία της εφαρμογής εφόσον αντικαθιστά στο μεγαλύτερο τμήμα της εφαρμογής τα Intents για την εσωτερική επικοινωνία.

Επιπλέον γίνεται μια σύντομη παρουσίαση της βιβλιοθήκης DragSortListView η οποία παρέχει έτοιμη λειτουργία drag-sort για λίστες αντικειμένων. Ακόμη παρουσιάζεται η βιβλιοθήκη Crouton η οποία συνιστά μια εναλλακτική προσέγγιση στα Toast messages του Android SDK, καθώς και η λειτουργία της βιβλιοθήκης Jackson ή οποία αναλαμβάνει την μετατροπή JSON σε αντικείμενα της Java.

3.1 Roboguice & Google Guice

Το Roboguice είναι μια βιβλιοθήκη που βασίζεται βιβλιοθήκη Google Guice και επεκτείνει τη λειτουργικότητα της για την χρήση με την πλατφόρμα Android. Δημιουργήθηκε από τον Michael Burton, και είναι δωρεάν διαθέσιμη ως λογισμικό ανοιχτού κώδικα κάτω από το Apache License version 2. Η βιβλιοθήκη παρέχει δυνατότητα χρήσης του dependency injection για την ρύθμισή των αντικειμένων της Java με τη χρήση annotations. Πριν προχωρήσουμε στο framework καλό θα ήταν να γίνει αρχικά μια αναφορά στο τι ακριβώς είναι και τι κάνει το dependency injection.

3.1.1 Dependency Injection

Όρος dependency injection επινοήθηκε από τον Martin Fowler στο άρθρο του «Inversion of Control Containers and the Dependency Injection Pattern». Το dependency injection είναι ένα σχεδιαστικό πρότυπο (design pattern) το οποίο επιτρέπει την αφαίρεση των ενσωματωμένων εξαρτήσεων (hard-coded dependencies) και καθιστά δυνατή την αλλαγή τους είτε κατά το χρόνο εκτέλεσης είτε κατά το χρόνο της μεταγλώττισής. Στην πραγματικότητα πρόκειται για ένα τρόπο παραμετροποίησης ενός αντικειμένου στο οποίο τα πεδία του αντικειμένου και τα συνεργαζόμενα αντικείμενα ορίζονται από μια εξωτερική οντότητα. Στο πλαίσιο της διαχείρισης των dependencies (dependency management) το αντικείμενο δεν έχει την ευθύνη της αρχικοποίησης των dependencies του, αντίθετα η ευθύνη αυτή περνά σε ένα άλλο εξωτερικό μηχανισμό, αντιστρέφοντας με τον τρόπο αυτό τον έλεγχο (Inversion of control).

Το IoC μεταφέρει της δευτερεύουσες ευθύνες από ένα αντικείμενο σε άλλα αντικείμενα τα οποία είναι στο συγκεκριμένο σκοπό, υποστηρίζοντας με τον τρόπο αυτό την Αρχή Μοναδικής Αρμοδιότητας (Single Responsibility Principle).

Το dependency injection περιλαμβάνει τρία βασικά στοιχεία, το τμήμα που καταναλώνει τα dependencies (depended consumer), τη δήλωση των dependencies ενός συστατικού (component), και τέλος τον injector (αλλιώς γνωστό και ως provider ή container). Ο injector δημιουργεί τα αντικείμενα των κλάσεων που ικανοποιούν κάποιο dependency κατά βούληση.

3.1.2 Injector

Ο injector είναι η εξωτερική οντότητα η οποία αναλαμβάνει την διαχείριση των dependencies. Έχει γνώση των dependencies, πιο concrete class για παράδειγμα ικανοποιεί τις απαιτήσεις κάποιου εξαρτώμενου (dependent) αντικειμένου και αναλαμβάνει να το παρέχει στο αντικείμενο. Ο ρόλος του injector είναι να αρχικοποιήσει (instantiate) και να εισάγει τα dependencies, όμως δεν σταματά εκεί μιας και συνήθως έχει περισσότερες αρμοδιότητες από αυτές.

Κατά τη ρύθμισή (configuration) του injector μπορούμε να ορίσουμε το ποια στοιχεία θα μπορεί να αρχικοποιήσει (instantiate), καθώς και το τι dependencies θα πρέπει να εισάγει σε κάθε στοιχείο. Επιπλέον μπορεί να οριστεί με ποιο τρόπο γίνετε η αρχικοποίηση κάθε στοιχείου, για παράδειγμα το εάν θα πρέπει να δημιουργείται ένα καινούργιο αντικείμενο κάθε φορά που ο injector εισάγει κάποιο dependency ή εάν θα πρέπει να χρησιμοποιείται το ίδιο αντικείμενο (singleton pattern).

Στην περίπτωση που κάποια στοιχεία έχουν οριστεί ως singleton μερικά containers έχουν τη δυνατότητα να καλέσουν μεθόδους στο singleton αντικείμενο κατά τον τερματισμό λειτουργίας του injector. Με τον τρόπο αυτό ένα singleton μπορεί να ελευθερώσει δεσμευμένους πόρους, όπως για παράδειγμα συνδέσεις δικτύου, ή συνδέσεις σε βάσεις δεδομένων. Το φαινόμενο αυτό συχνά αποκαλείται «διαχείριση του κύκλου ζωής του αντικειμένου» (instance life cycle management). Αυτό ουσιαστικά σημαίνει ότι ο injector έχει τη δυνατότητα να διαχειριστεί το συστατικό (component) στα διάφορα στάδιο του κύκλου ζωής του (π.χ. δημιουργία, αρχικοποίηση, ρύθμιση και καταστροφή).

Η διαχείριση του κύκλου ζωής είναι μια από τις ευθύνες του dependency container σε συνδυασμό με την δημιουργία και την εισαγωγή (injection). Λόγω του ότι το container συνήθως διατηρεί αναφορές σε στοιχεία μετά την δημιουργία τους ονομάζεται container και όχι factory (factory pattern). Το container κατά κύριο λόγο κρατά αναφορές μόνο σε αντικείμενα για τα οποία διαχειρίζεται τον κύκλο ζωής ή τα οποία πρόκειται να επαναχρησιμοποιηθούν όπως singletons ή flyweights. Στην περίπτωση που έχει ρυθμιστεί να δημιουργεί καινούργια αντικείμενα για μερικά από τα components σε κάθε κλήση στον injector, αυτός συνήθως δεν διατηρεί αναφορές στα αντικείμενα που έχουν δημιουργηθεί (θα μπορούσαμε να πούμε κατά κάποιο τρόπο ξεχνά τη δημιουργία τους). Αυτό συμβαίνει για να αποφευχθεί πρόβλημα κατά την συλλογή των αντικειμένων (λόγω αναφορών σε αυτά από στον dependency injector) από τον garbage collector μετά το τέλος της χρήσης τους.

3.1.3 Τύποι DI

Σύμφωνα με τον Martin Fowler υπάρχουν τρεις διαφορετικοί τύποι με τους οποίους ένα αντικείμενο μπορεί να αποκτήσει μια αναφορά σε μια εξωτερική οντότητα, ανάλογα με το πρότυπο χρησιμοποιείται για να παρέχει την εξάρτηση. Ο πρώτος τύπος είναι το interface injection, όπου η εξαγόμενη μονάδα παρέχει ένα interface το οποίο θα πρέπει να υλοποιηθεί από τους χρήστες του, ώστε να μπορεί να πάρει τα dependencies του κατά το χρόνο εκτέλεσης. Ο δεύτερος τύπος ονομάζεται setter injection, στον οποίο η εξαρτώμενη μονάδα παρέχει μια setter μέθοδο την οποία το framework χρησιμοποιεί για να εισάγει την εξάρτηση. Ο τρίτος τύπος ονομάζεται constructor injection κατά τον οποίο οι εξαρτήσεις εισάγονται μέσο του constructor της κλάσης. Να σημειωθεί ότι ανάλογα το framework μπορεί να υπάρχουν επιπλέον τύποι.

3.1.4 Πλεονεκτήματα και μειονεκτήματα

Η δυνατότητα να αποφασίσουμε τα ποια dependencies θα εισάγουμε κατά το χρόνο εκτέλεση, αντί για το χρόνο μεταγλώττισης είναι ένα αρκετά σημαντικό πλεονέκτημα. Για παράδειγμα διαφορετικές υλοποιήσεις του ίδιου συστατικού μπορούν να δημιουργηθούν κατά το χρόνο εκτέλεσης και να εισαχθούν στον ίδιο δοκιμαστικό (test) κώδικα. Έτσι μπορεί να γίνει δοκιμή της κάθε διαφορετικής υλοποίησης χωρίς όμως ο δοκιμαστικός να γνωρίζει ότι υπήρχε η οποιαδήποτε αλλαγή στην υλοποίηση.

Το dependency injection έχει ως κύριο στόχο του να επιτρέψει την επιλογή πολλαπλών υλοποιήσεων για κάποιο dependency, κατά το χρόνο εκτέλεσης ή με την χρήση αρχείων ρυθμίσεων αντί για το σύνηθες που είναι τα dependencies να καθορίζονται κατά την μεταγλώττιση. Είναι ιδιαίτερα χρήσιμο κατά την δοκιμή (testing) πολύπλοκων συστατικών (components) με την χρήση εικονικών υλοποιήσεων δοκιμής (test stubs).

Το unit testing συστατικών σε μεγάλα συστήματα παρουσιάζει κάποιο βαθμό δυσκολίας λόγω του ότι τα συστατικά αυτά συνήθως απαιτούν ένα μεγάλο αριθμό υποδομής και ρύθμισης ώστε να λειτουργήσουν. Με την χρήση του dependency injection η διαδικασία της αρχικοποίησης ενός μεμονωμένου λειτουργικού συστατικού ώστε αυτό να αυτό να γίνει δοκιμή του απλοποιείται. Λόγω του ότι τα συστατικά απλά δηλώνουν τα dependencies τους ένα test έχει την δυνατότητα απλά να αρχικοποιήσει μόνο τα απολύτως απαραίτητα ώστε να εκτελεστεί το test.

Είναι σημαντικό να σημειωθεί ο injector παρέχει την δυνατότητα να αντικαταστήσει διάφορα συστατικά με απλοποιημένες υλοποιήσεις τους (stub implementations) για το test, επιτρέποντας την απομονωμένη δοκιμή συστατικών.

Το dependency injection προσπαθεί να αποσυνδέσει τον κώδικα εξαλείφοντας ή μειώνοντας σημαντικά τα περιττά dependencies μεταξύ των αντικειμένων. Ως γνωστόν τα διάφορα συστατικά (components) είναι αρκετά ευάλωτα στις αλλαγές που συμβαίνουν στα dependencies τους, οπότε αν συμβεί η οποιαδήποτε αλλαγή σε κάποιο από αυτά θα πρέπει και το ίδιο να προσαρμοστεί στις αλλαγές αυτές.

Κάνοντας χρήση του dependency injection ξεφορτωνόμαστε επίσης ένα μεγάλο αριθμό από κλήσεις σε static μεθόδους των factories. Το testing των

factories είναι δυσκολότερο διότι δεν είναι εφικτό να τροποποιηθεί η πραγματική τους συμπεριφορά όπως γίνεται με τα interfaces.

Επιπλέον μειώνοντας τον αριθμό των διαφορετικών dependencies καθίσταται εφικτή η επαναχρησιμοποίηση συστατικών σε διαφορετικό πλαίσιο (context). Εφόσον έχουμε εξωτερική εισαγωγή των dependencies μπορούμε για παράδειγμα να χρησιμοποιήσουμε το ίδιο συστατικό με μια διαφορετική υλοποίηση κάποιου interface, ή να χρησιμοποιήσουμε το συστατικό σε κάποιο άλλο πλαίσιο χωρίς να προβούμε σε κάποια τροποποίηση του κώδικα του.

Λόγω του ότι τα dependencies των οποίων η διαχείριση γίνεται από τον injector είναι κατά κάποιο τρόπο μαύρα κουτιά όσον αφορά το υπόλοιπο σύστημα, αυτό καθιστά δυσκολότερη την ανακάλυψη και την λύση πιθανών προβλημάτων. Επιπλέον λόγο το ότι ουσιαστικά το dependency injection αποκρύπτει τα dependencies μεταξύ των κλάσεων, πολλά προβλήματα τα οποία προκαλούνται είτε από ελλιπείς υλοποιήσεις, είτε λόγο μη διαθέσιμων dependencies δεν είναι προφανή κατά το χρόνο μεταγλώττισης. Τα προβλήματα αυτά συνήθως εμφανίζονται ως σφάλματα κατά το χρόνο εκτέλεσης. Επίσης σε κάποιες περιπτώσεις η χρήση του dependency injection container μπορεί να κάνει δυσκολότερη τη συντήρηση του κώδικα επειδή δεν είναι πάντοτε προφανές το πότε μια αλλαγή σε κάποιο σημείο του κώδικα μπορεί να προκαλέσει προβλήματα.

3.1.5 Roboguice

Για τη χρήση του `Roboguice` κατά την ανάπτυξη της εφαρμογής απαιτείται από τον προγραμματιστή να επεκτείνει τις αντίστοιχες `Robo*` classes. Αντί για παράδειγμα να κληρονομήσουμε απευθείας από την class Activity για να χρησιμοποιήσουμε το `Roboguice` θα πρέπει η sub class μας να κληρονομεί από την `RoboActivity` ή οποία είναι subclass της `Activity` με ενσωματωμένη τη λειτουργικότητα του `RoboGuice`. Αντίστοιχα υπάρχουν `Robo*` classes για `Fragments`, `ListFragments` κ.λπ. Αν παρατηρήσει κανείς τον κώδικα θα δει ότι στην πραγματικότητα κληρονομούμε από τις `RoboSherlock*` classes, όπως έχει αναφερθεί αυτό συμβαίνει λόγο της χρήσης του `ActionBarShelock`. Οι `RoboSherlock*` classes είναι αντίστοιχες με τις `Robo*` classes με τη μόνη διαφορά ότι κληρονομούν από τις `Sherlock*` classes αντί για τις classes του SDK. (Αξίζει να σημειωθεί ότι για την έκδοση 3 η οποία είναι επόμενη του Roboguice υπό ανάπτυξη θα υπάρχει άμεση υποστήριξη για την χρήση της ActionBar).

Το `Roboguice` όπως και το `Guice` χρησιμοποιεί annotations για την επίτευξη του dependency injection. Αρχικά έχουμε το annotation `@Inject` το οποίο χρησιμοποιείται για να κάνουμε inject αντικείμενα απευθείας στον κώδικα. Το συγκεκριμένο annotation μπορεί να χρησιμοποιηθεί για να κάνει inject σε constructors, μεθόδους και πεδία (μεταβλητές). Εάν παρατηρήσουμε την κάποια από τις `Robo*` classes θα δούμε ότι κατά την `onCreate()` γίνεται κλήση του injector ο οποίος εισάγει τα αντικείμενα στα αντίστοιχα fields.

Αντίστοιχα το Roboguice παρέχει την δυνατότητα για την εισαγωγή Views με την χρήση του annotation `@InjectView`. Για παράδειγμα μπορούμε να κάνουμε inject ένα `TextView` με τη παρακάτω σήμανση.

```
@InjectView(R.id.menuConnector) TextView menuConnector;
```

Η συγκεκριμένη κλήση έχει το ίδιο αποτέλεσμα με την κλήση της μεθόδου `findViewById()`.

```
 TextView menuConnector = (TextView) findViewById(R.id.name)
```

Κοιτώντας τις `Robo*` classes θα δούμε ότι γίνεται η κλήση της μεθόδου για το injection `onViewCreated()` για τα fragments και στην `onContentChanged()` για τα activities πάντα δηλαδή αφού έχει δημιουργηθεί το content του view από το xml layout. Υπήρχαν περιπτώσεις κατά το στάδιο της ανάπτυξης που το view injection δεν δούλεψε όπως θα έπρεπε, η κλήση της `onContentChange()` πραγματοποιήθηκε πριν από τον πραγματικό ορισμό του content, με αποτέλεσμα όπως είναι αναμενόμενο να υπάρχουν null pointer exceptions.

Επιπλέον το `RoboGuice` παρέχει την δυνατότητα για resource injection με τη χρήση του annotation `@InjectResource`, όπως φαίνεται παρακάτω:

```
@InjectResource(R.anim.my_animation) Animation myAnimation;
```

το οποίο παρέχει την δυνατότητα να κάνουμε inject σε μεταβλητές τα διάφορα Android resources.

Επιπλέον το `RoboGuice` μέσω του `DefaultRoboModule` το οποίο περιέχει κάποιο βασικό configuration στο `Guice` ώστε να μπορεί να κάνει inject βασικά τμήματα (για παράδειγμα system services) του Android. Έτσι ο προγραμματιστής έχει για παράδειγμα τη δυνατότητα να κάνει inject το `WifiManager` του Android με τη χρήση του annotation χωρίς να απαιτείται κάτι επιπλέον από τον ίδιο.

Εφόσον το framework αναλαμβάνει την δημιουργία και την διαχείριση των αντικειμένων μας παρέχει κι ένα εύκολο τρόπο για τη δημιουργία singleton. Αντί για τη χρήστη του παραδοσιακού τρόπου ο οποίος απαιτεί την χρήση μιας static μεθόδου η οποία μας επιστρέφει το single instance μιας class, μπορούμε να χρησιμοποιήσουμε το `@Singleton`.

```
@Singleton
public class Controller extends RoboService {...}
```

Με τον τρόπο αυτό το framework γνωρίζει ότι τη συγκεκριμένη class είναι singleton και κάθε φορά που αυτή θα ζητηθεί από τον injector θα εισάγει πάντα το ίδιο ένα instance. Η χρήση του annotation δεν είναι ο μόνος τρόπος για την φυσικά, το ίδιο πράγμα μπορεί να γίνει με διάφορους τρόπους, ένα από τους οποίους μπορούμε να δούμε στην παράγραφο για το Otto event bus.

Στην περίπτωση που μας είναι απαραίτητο να προσθέσουμε πρόσθετους κανόνες (bindings) πέρα από αυτούς που παρέχει το `RoboGuice` τότε το πρώτο βήμα είναι να επεκτείνουμε την class `AbstractModule` ένα παράδειγμα μπορούμε να δούμε στην class `RemoteModule`.

```
public class RemoteModule extends AbstractModule {
    @Override public void configure() {
        bind(Bus.class).toInstance(new Bus(ThreadEnforcer.ANY, "mbrcbus"));
        bind(ObjectMapper.class).toInstance(new ObjectMapper());
    }
}
```

Η `AbstractModule` περιέχει τη μέθοδο `configure()` την οποία θα πρέπει να κάνουμε override. Μέσα σε αυτή παρέχουμε τα διάφορα bindings που θέλουμε. Η χρήση της μεθόδου `toInstance()` μεταφέρει την ευθύνη για την απόκτηση του αντικειμένου από το Guice στο module. Κατά τη δημιουργία του ο injector θα πραγματοποιήσει αυτόματα injection στα πεδία και τις μεθόδους του αντικειμένου αλλά αν συναντήσει έναν injectable constructor θα τον αγνοήσει.

Στη συνέχεια απαιτείται να δηλώσουμε το module ώστε το RoboGuice να γνωρίζει την ύπαρξη του.

```
RoboGuice.setBaseApplicationInjector(this, Stage.PRODUCTION,
   Modules.override(RoboGuice.newDefaultRoboModule(this))
              .with(new RemoteModule()));
```

Η μέθοδος `override()` χρησιμοποιείται για τη δημιουργία ενός νέου module το οποίο ενώνει υπάρχοντα modules επικαλύπτοντας τα κοινά bindings. Έτσι για παράδειγμα αν τα ίδια bindings είναι υπαρκτά και στα δυο modules τα bindings του πρώτου επικαλύπτουν αυτά του δεύτερου module.

Η δήλωση του module γίνεται μέσα στην class `RemoteApplication` η οποία επεκτείνει την class `Application`. Σημαντικό είναι να θυμηθούμε ότι εφόσον έχουμε μια δική μας υλοποίηση της `Application` αυτή θα πρέπει επίσης να δηλωθεί και στο `AndroidManifest.xml` κάτω από το tag `<application>` με το attribute `android:name`.

Φυσικά καλό θα ήταν να σημειωθεί ότι τα στοιχεία που παρουσιάζονται δεν είναι παρά ένα μικρό τμήμα των δυνατοτήτων του framework, και αποτελούν κυρίως τμήματα που χρησιμοποιήθηκαν κατά την ανάπτυξη της εφαρμογής.

3.2 Otto Event Bus

Το Otto είναι ένα event bus που έχει σχεδιαστεί με στόχο να αποσυνδέσει τα διάφορα τμήματα της εφαρμογής διατηρώντας ταυτόχρονα την δυνατότητα για αποτελεσματική επικοινωνία. Βασίζεται στη βιβλιοθήκη Guava της Google αλλά προσθέτει επιπλέον λειτουργικότητα ειδικά για το Android.

```
Bus bus = new Bus();
```

Η επικοινωνία μέσω του Otto γίνεται μέσα από ένα αντικείμενο της class Bus. Το bus είναι αποτελεσματικό όταν το διαμοιράζονται όλες οι class που πρέπει να επικοινωνήσουν. Αν κοιτάξουμε την `RemoteModule` θα δούμε ότι υπάρχει η παρακάτω ρύθμιση που απευθύνεται στον injector:

```
bind(Bus.class).toInstance(new Bus(ThreadEnforcer.ANY, "mbrcbus"));
```

Αύτη η εντολή ενημερώνει ζητά από τον injector κάθε φορά που του ζητείται ένα instance της class `Bus` να χρησιμοποιεί το ίδιο ένα instance της `Bus` (θα μπορούσαμε να πούμε ότι κατά κάποιο τρόπο έχουμε το Singleton Pattern στην περίπτωση όμως αυτή δεν ακολουθείται το γνωστό pattern, αλλά αυτό εξασφαλίζεται από το Dependency Injection framework). Η παράμετρος `ThreadEnforcer.ANY` ενημερώνει το event bus ότι επιτρέπετε η επικοινωνία μεταξύ

διαφορετικών thread (π.χ. ui thread και background thread), η προεπιλεγμένη συμπεριφορά περιορίζει τα μηνύματα στο κύριο thread.

Το event publishing είναι το σημαντικότερο χαρακτηριστικό του bus μιας και επιτρέπει την ενημέρωση των subscribers για κάποιο γεγονός που έχει συμβεί. Μπορεί να σταλεί ένα αντικείμενο μιας οποιασδήποτε κλάσης στο bus και θα αποσταλεί μόνο στους subscribers για τον συγκεκριμένο τύπο (κλάση). Το publish γίνεται με τη χρήση της μεθόδου `post`.

```
public void setRating(double rating) {
  this.rating = (float) rating;
  bus.post(new RatingChanged(this.rating));
}
```

Το subscription είναι το συμπλήρωμα του publishing (ενημερώνει ότι κάποιο γεγονός έχει συμβεί). Για να γίνει μια μέθοδος subscriber για ένα event αρκεί να χρησιμοποιήσει τη σήμανση `@Subscribe`. Η μέθοδος θα πρέπει να έχει μόνο μια παράμετρο, της οποίας ο τύπος θα πρέπει να είναι η κλάση του event το οποίο μας ενδιαφέρει.

```
@Subscribe public void handleRatingChange(RatingChanged event) {
  if (trackRating != null) {
   trackRating.setRating(event.getRating());
  }
}
```

Το όνομα της μεθόδου δεν έχει καμία σημασία, αλλά απαιτείται η σήμανση, η μοναδική παράμετρος καθώς και η δημόσια (public) πρόσβαση.

Για να μπορεί να λάβει τα events ένα αντικείμενο χρειάζεται να κάνει register στο bus. Αυτό μπορεί να γίνει με τη χρήση της μεθόδου `register`:

```
bus.register(this);
```

Αντιστοίχως μετά τη χρήση της μεθόδου `unregister` το αντικείμενο θα σταματήσει να λαμβάνει events. Αυτό είναι πολύ χρήσιμο για παράδειγμα κατά τον κύκλο ζωής ενός fragment όπου η κλάση κάνει `register` όταν εμφανίζεται στην οθόνη και `unregister` όταν κρύβεται.

Αξίζει να σημειωθεί ότι το Otto θα βρει μόνο τις μεθόδους της άμεσης κλάσης και θα αγνοήσει ότι υπάρχει σε τυχών super classes ή interfaces τα οποία μπορεί να υλοποιεί η τρέχουσα κλάση.

Ένα ακόμη σημαντικό χαρακτηριστικό του Otto είναι οι producers. Οι producers χρησιμεύουν ώστε να ενημερώνουν με τρέχουσες γνωστές τιμές για συγκεκριμένα γεγονότα. Ένας producer καλείται ακριβώς μετά το registration και ενημερώνει τους subscribers.

Η δημιουργία ενός producer γίνεται με τη χρήση της σήμανσης `@Produce`. Η μέθοδος δεν δέχεται παραμέτρους και επιστρέφει τον τύπο του event που μας ενδιαφέρει. Οι Producers όπως και οι Subscribers πρέπει να κάνουν register.

```
@Produce public RatingChanged produceRatingChanged() {
  return new RatingChanged(this.rating);
}
```

Καλό θα ήταν να σημειωθεί ότι δεν μπορεί να γίνει άμεσα post ενός event από κάποιο background thread στο ui thread. Ο καταλληλότερος τρόπος για να γίνει κάτι τέτοιο είναι με τη δημιουργία ενός wrapper ο οποίος ελέγχει αν το τρέχον message loop είναι το βασικό κι αν δεν είναι χρησιμοποιεί το `Handler` για να περάσει το event στο βασικό thread.

```
public void post(final Object event) {
  if (Looper.myLooper() == Looper.getMainLooper()) {
   bus.post(event);
  } else {
   mHandler.post(new Runnable() {
       @Override public void run() {
             bus.post(event);
       }
   });
  }
}
```

3.3 DragSortListView

Η `DragSortListView` είναι μια βιβλιοθήκη η οποία επεκτείνει την `ListView` του Android και προσθέτει λειτουργικότητα η οποία επιτρέπει την αναδιοργάνωση τον αντικειμένων της λίστας με drag-and-drop.

Για τη χρήση του DragSortListView απαιτείται να δηλώσουμε αρχικά το στοιχείο στο xml layout όπως φαίνεται παρακάτω.

```
<com.mobeta.android.dslv.DragSortListView
  android:id="@android:id/list"
  android:layout_width="match_parent"
  android:layout_marginTop="@dimen/side_margin_half"
  android:layout_marginBottom="@dimen/side_margin_half"
  android:layout_marginRight="@dimen/side_margin"
  android:layout_marginLeft="@dimen/side_margin"
  android:layout_height="0dip"
  android:layout_weight="1"
  android:drawSelectorOnTop="true"
  android:dividerHeight="@dimen/side_margin_half"
  android:divider="@null"
  android:fastScrollEnabled="true"
  android:listSelector="@drawable/abs__list_selector_holo_dark"/>
```

Βασικό τμήμα της βιβλιοθήκης είναι ο `DragSortController`. Ο controller υλοποιεί όλη εκείνη τη λειτουργικότητα η οποία είναι απαραίτητη για την εκκίνηση ενός drag ή για την αφαίρεση ενός dragged αντικειμένου (`ListItem`). Ο controller υλοποιεί το interface `View.OnTouchListener` για να επεξεργάζεται τα touch events που αποστέλλονται στο `DragSortListView`. Ο controller υλοποιεί επίσης το interface `FloatViewManager` μέσω της subclass `SimpleFloatViewManager` η οποία διαχειρίζεται τη δημιουργία του floating view. Στην περίπτωση που το `DragSortListView` δηλωθεί απευθείας στο layout με την χρήση των xml attributes δημιουργείται ένας default controller, σε αντίθετη περίπτωση θα πρέπει να δημιουργηθεί ένα αντικείμενο προγραμματιστικά και να περαστεί στις μεθόδους `setFloatViewManager()` και `setOnTouchListener()` του αντικειμένου της `DragSortListView`.

```
public DragSortController buildController(DragSortListView dslv) {
  DragSortController controller = new DragSortController(dslv);
  controller.setDragHandleId(R.id.drag_handle);
  controller.setFlingHandleId(R.id.click_remove);
  controller.setRemoveEnabled(removeEnabled);
  controller.setSortEnabled(sortEnabled);
  controller.setDragInitMode(dragStartMode);
  controller.setRemoveMode(removeMode);
  return controller;
}
```

Η μέθοδος `setDragHandleId()` ορίζει το id του στοιχείο που είναι υπεύθυνο για την εκκίνηση του drag, ουσιαστικά για να ξεκινήσει το drag θα πρέπει να πατήσουμε και να σύρουμε πάνω από το στοιχείο του οποίο το id έχουμε περάσει στη μέθοδο. Αντίστοιχα η `setFlingHandleId()` έχει παρόμοια λειτουργικότητα όσον αφορά την αφαίρεση στοιχείων από τη λίστα. Οι μέθοδοι `setRemoveEnabled()` και `setSortEnabled()` ενεργοποιούν τη δυνατότητα αφαίρεσης και αναδιάταξης των αντικειμένων τις λίστας αντίστοιχα. Τέλος οι μέθοδοι `setDragInitMode()` και `setRemoveMode()` ορίζουν τον τρόπο με τον οποίο γίνετε η εκκίνηση των drag και remove events.

Επιπλέον το `DragSortListView` απαιτεί τη χρήση τον ορισμό κάποιων event listeners. Έχουμε για παράδειγμα στη διάθεση μας το `DragSortListView.DropListener` όπως ορίζεται από το `DropListener` interface, το οποίο μας παρέχει τη μέθοδο `drop(int from, int to)`. Η μέθοδος εκτελείται όταν ο χρήστης σύρει και αφήσει ένα αντικείμενο σε κάποιο σημείο, οι παράμετροι from και το αντιπροσωπεύουν τον αύξον αριθμό (index) της αρχικής και της τελικής θέσης. Η σύνδεση της υλοποίησης του listener γίνεται περνώντας το αντικείμενο στην κλήση της μεθόδου `setDropListener()` του αντικειμένου της `DragSortListView`. Αντίστοιχα υπάρχουν listeners για το drag event και για το remove event, τα οποία πρέπει να υλοποιηθούν για την διαχείριση των αντίστοιχων events.

3.4 Crouton

Ένας τρόπος να στείλει η εφαρμογή ένα γρήγορο μήνυμα στο χρήστη είναι μέσω της χρήσης της class Toast. Το toast είναι ένα view που περιέχει ένα σύντομο μικρό μήνυμα για το χρήστη. Το toast view εμφανίζεται σαν να επιπλέει πάνω από την εφαρμογή. Ο σχεδιασμός του είναι τέτοιος ώστε να μην εμποδίζει τις δραστηριότητες του χρήστης σε μια προκειμένη στιγμή (π.χ. πληκτρολόγηση κάποιου κειμένου), οπότε δεν γίνεται ποτέ εστίαση σε αυτό. Το πρόβλημα με το Toast είναι ότι δεν γνωρίζει το τρέχον πλαίσιο εκτέλεσης, το οποίο σημαίνει ότι είναι κάλλιστα πιθανό το μήνυμα να εμφανιστεί αφού ο χρήστης έχει φύγει από την εφαρμογή. Το crouton λύνει αυτό το πρόβλημα δημιουργώντας ένα τρόπο παρουσίασης μηνυμάτων στο χρήστη λαμβάνοντας υπόψιν το γενικό πλαίσιο (context) εκτέλεσης της εφαρμογής.

Το crouton εμφανίζει το μήνυμα σε μια θέση η οποία επιλέγει ο χρήστης, συνήθως κάτω από το σημείο της ActionBar. Τα μηνύματα μπορούν να

τοποθετηθούν σε κάποιο είδος ουράς μηνυμάτων (message queue) και να εμφανιστούν το ένα μετά το άλλο.

Εικόνα 13. Crouton Styles

Όλο το παρουσιαστικό κομμάτι του crouton ορίζεται από το `Style`, το crouton έχει τρία βασικά `Style` όπως φαίνεται στην Εικόνα 13. Αυτά είναι τα `Style.ALERT`, `Style.CONFIRM` και `Style.INFO`, όμως ο χρήστης μπορεί να ορίσει το δικό του Style. Σε ένα Style μπορούμε να ορίσουμε τη χρονική διάρκεια την οποία θα είναι εμφανές το μήνυμα, το πλάτος και ύψος του, το animation που χρησιμοποιεί κατά την εμφάνιση του και την εξαφάνιση του κ.α.

Η δημιουργία και η εμφάνιση ενός crouton γίνεται με απλό τρόπο, αρκεί να καλέσουμε τη μέθοδο `Crouton.makeText(Activity, CharSequence, [Style]).show();` Η μέθοδος δέχεται ως παραμέτρους μια αναφορά στο τρέχον Activity (context), ένα `CharSequence` το οποίο είναι το μήνυμα που θέλουμε να εμφανίσουμε και τέλος το style με το οποίο θα πρέπει να εμφανιστεί το crouton. Αντίστοιχα υπάρχει `Crouton.makeText(Activity, int, Style).show();` Η οποία δέχεται το resource id από το string το οποίο θα πρέπει να εμφανίσει στη θέσης του `CharSequence`.

Λόγω του ότι υπάρχει ένα πρόβλημα με την βιβλιοθήκη κατά την αλλαγή του Orientation απαιτείται να γίνεται κλήση της μεθόδου `Crouton.cancelAllCroutons();` για να ακυρώσει όλα τα προγραμματισμένα Croutons όταν καταστρέφετε το `Activity` (`Activity.onDestroy()`).

3.5 Jackson

Το Jackson είναι μια βιβλιοθήκη η οποία κάνει parsing JSON (JavaScript Object Notation) και το μετατρέπει σε μορφή που μπορεί να χρησιμοποιηθεί μέσα στο πρόγραμμα (μεταβλητές, και ιδιότητες με τιμές).

Υποστηρίζει τρεις βασικούς τρόπους επεξεργασίας, από τους οποίους είναι το data binding. Το data binding είναι ο ευκολότερος τρόπος χρήσης και επιτρέπει την απρόσκοπτη μετατροπή μεταξύ JSON και αντικειμένων της Java. Το data binding χρησιμοποιεί το Streaming API ως βασικό σύστημα για την ανάγνωση και την εγγραφή του JSON. Με τον τρόπο αυτό επιτυγχάνει υψηλή απόδοση (συγκριτικά με τις υπόλοιπες data binding βιβλιοθήκες), αλλά προσθέτει κάποια επιπλέον επιβάρυνση ιδιαίτερα όταν συγκρίνεται με τις καθαρά streaming βιβλιοθήκες.

Το Jackson παρέχει δύο διαφορετικούς τρόπους για data binding. Ο πρώτος τρόπος χρησιμοποιεί τους καθιερωμένους container τύπους του JDK (List, Map) και τους μονοδιάστατους τύπους (scalar) (String, Boolean, Number, nulls). Ο δεύτερος τρόπος είναι το πλήρες data binding το οποίο μπορεί να χρησιμοποιήσει μια ευρεία γκάμα από τύπους της Java συμπεριλαμβανομένων των «POJO»

(Plain Old Java Objects). Και οι δύο τύποι χρησιμοποιούν το αντικείμενο `ObjectMapper`.

Δεν υφίσταται καμία διαφορά κατά το serialization (δημιουργία JSON από αντικείμενα της Java) όσον αφορά τους δύο παραπάνω τρόπος, οι διαφορές αυτές έχουν σημασία μόνο κατά την ανάγνωση και την μετατροπή σε αντικείμενα της java. Το serialization μπορεί να γίνει με τον παρακάτω τρόπο.

```
ObjectMapper mapper = new ObjectMapper();
mapper.writeValue(dst, obj);
```

Το `dst` μπορεί να είναι κάποιο `File`, `OutputStream` ή `Writer`, και το `obj` είναι ένα Java αντικείμενο το οποίο θέλουμε να μετατρέψουμε σε JSON. Επίσης υπάρχει και η μέθοδος `mapper.writeValueAsString(obj)` στο οποίο μπορούμε να περάσουμε ένα αντικείμενο της java και να πάρουμε το serialized JSON string.

Όταν χρησιμοποιούμε το full data binding ο τύπος για το deserialization πρέπει να έχει πλήρως οριστεί για παράδειγμα έχουμε την κλάση `SocketMessage`, με τρεις ιδιότητες, συμπεριλαμβανομένων των getter και setter function για αυτές.

```
public class SocketMessage {
    @JsonProperty private String context;
    @JsonProperty private String type;
    @JsonProperty private Object data;
  public SocketMessage() {
  ...
  }
  ...
}
```

Το full data binding θα διαβάσει ένα JSON string της ακόλουθης μορφής και θα δημιουργήσει ένα `SocketMessage` αντικείμενο το οποίο θα περιέχει τις τιμές που θα διαβαστούν από το JSON.

```
{"context":"","type":"", "data":""}
```

Με τη χρήση της μεθόδου `readValue()` του `ObjectMapper` μπορούμε να διαβάσουμε το μήνυμα από το `src` και να το περάσουμε τις τιμές στο αντίστοιχο αντικείμενο.

```
SocketMessage msg = mapper.readValue(src, SocketMessage.class)
```

Πέρα από το full data binding υπάρχει και το απλό data binding όπως αναφέρθηκε το οποίο είναι περιορισμένο στους βασικούς τύπους του JDK. Στον Πίνακας 1 μπορούμε να δούμε τους τύπους που χρησιμοποιούνται στο απλό data binding. Το JSON Type αναφέρετε στους τύπους ιδιοτήτων του JSON ενώ το Java Type αναφέρετε στους τύπους του JDK στους οποίους θα τοποθετηθούν οι αυτές οι τιμές με τη χρήση του απλού data binding.

JSON Type	Java Type
`Object`	`LinkedHashMap<String,Object>`
`Array`	`ArrayList<Object>`
`string`	`String`

number (no fraction)	Integer, Long ή BigInteger (το μικρότερο δυνατό)
number (fraction)	Double (ρυθμιζόμενο ώστε να χρησιμοποιεί BigDecimal)
true|false	Boolean
Null	null

Πίνακας 1. JSON type to Java type

Αν είναι αποδεκτός ο περιορισμός στους βασικούς τύπους του JDK τότε το deserialization μπορεί να γίνει απλά με τη χρήση του `Object.class`.

Ένας από τους άλλους τρόπους επεξεργασίας που υποστηρίζει το Jackson είναι το Tree Model. Το tree model σε εννοιολογικό επίπεδο μοιάζει αρκετά με το tree model του DOM XML, παρόλο που υπάρχουν αρκετές διαφορές λόγο δομικών και σημασιολογικών διαφορών μεταξύ του JSON και του XML.

Η κατασκευή των δέντρων γίνεται με τη χρήση του `ObjectMapper`, με παρόμοιο τρόπο με το data binding.

```
JsonNode node = mapper.readValue(reply, JsonNode.class);
```

Το interface `JsonNode` παρέχει μεθόδους για την πρόσβαση (ανάγνωση) στις πληροφορίες του node. Για τη βασική διάσχιση του δέντρου υπάρχει διαφοροποίηση μεταξύ των μεθόδων που αφορούν τα αντικείμενα (JSON Objects) και αυτών που αφορούν τους πίνακες (JSON Arrays). Στα αντικείμενα η πρόσβαση επιτυγχάνεται με τη χρήση του ονόματος της ιδιότητας ενώ στους πίνακες με τον αύξων αριθμό του στοιχείου (index). Επιπροσθέτως υπάρχει η δυνατότητα χρήσης «ασφαλών» μεθόδων οι οποίες επιστρέφουν ένα `MissingNode` αντικείμενο αντί για null όταν ένα αντικείμενο ή πίνακας δεν περιέχει κάποια τιμή.

Από τις μεθόδους που μας παρέχει το `JsonNode` έχουμε την `get(int index)` η οποία επιστρέφει το στοιχείο index, και την `get(String property)` η οποία χρησιμοποιείται για αντικείμενα και μας επιστρέφει την τιμή τις ιδιότητας με όνομα property. Η μέθοδος `path(int index)`, `path(String property)` έχει την ίδια λειτουργία με τις αντίστοιχες get με τη διαφορά ότι επιστρέφει `MissingNode` στη θέση του null.

Το `JsonNode` παρέχει επίσης μεθόδους για την ανάγνωση της τιμής από ένα node όπως (`asBoolean()`, `asDouble()` κ.α.) οι οποίες επιστρέφουν μια τιμή στον αντίστοιχο τύπο.

Επίλογος

Στο κεφάλαιο αυτό είδαμε τις βιβλιοθήκες που χρησιμοποιήθηκαν κατά την ανάπτυξη και κάποιες από τις τεχνικές οι οποίες γίνονται εφικτές χάρη στην χρήση κάποιων από τις βιβλιοθήκες. Έγινε μια σύντομη παρουσίαση του Dependency Injection pattern καθώς και της χρήσης του Dependency Injection framework RoboGuice το οποίο επεκτείνει τις λειτουργίες του Google Guice για την πλατφόρμα Android, επιπλέον είδαμε τη βασική χρήση του Otto event bus που χρησιμοποιείται για την εσωτερική επικοινωνία στην εφαρμογή καθώς και τη χρήση των υπόλοιπων βιβλιοθηκών που χρειαστήκαν για την υλοποίηση της εφαρμογής.

Στο επόμενο κεφάλαιο παρουσιάζονται πράγματα γύρω από την ανάπτυξη και το σχεδιασμό της εφαρμογής.

ΚΕΦΑΛΑΙΟ 4

ΑΝΑΠΤΥΞΗ

Εισαγωγή

Το είδος της εφαρμογής επιλέχθηκε αρχικά για να καλύψει προσωπικές ανάγκες, δηλαδή να παρέχει τη δυνατότητα να γίνει απομακρυσμένη διαχείριση της αναπαραγωγής μουσικής στον υπολογιστή μέσω της Android συσκευής η οποία βρίσκεται συνδεδεμένη σε ένα Wi-Fi δίκτυο. Φυσικά πέρα από αυτό στόχος ήταν να γίνει δωρεάν διάθεση της ώστε να καλύψει τις ανάγκες επιπλέον χρηστών, μιας και δεν υπήρχε κάτι διαθέσιμο για το συγκεκριμένο πρόγραμμα αναπαραγωγής (MusicBee).

Κατά την ανάπτυξη της η εφαρμογή υπήρξε πεδίο πειραματισμού γύρω από τεχνολογίες και τεχνικές από το χώρο της Java αλλά και της μηχανικής λογισμικού. Παράλληλα κατά το σχεδιασμό και την ανάπτυξη υπήρξαν εφαρμογές και βιβλιοθήκες οι οποίες υπήρξαν πηγή έμπνευσης όσο προς τη σχεδιαστική κατεύθυνση, τόσο την οπτική όσο και από πλευράς υλοποίησης, της εφαρμογής. Μεταξύ αυτών είναι η εφαρμογή Apollo η οποία είναι η επίσημη εφαρμογή μουσικής του CyanogenMod, η εφαρμογή Google Play Music καθώς και το Robotlegs framework για ActionScript 3.0. καθώς και το framework PureMVC για Java.

Η εφαρμογή είναι σχεδιασμένη ακολουθώντας ιδέες που δανείζονται στοιχεία από διάφορα design patterns συμπεριλαμβανομένων των Model View Controller, Command Pattern και τα συνδυάζουν με τη χρήση του Dependency Injection pattern. Επίσης η χρήση του event bus έπαιξε και αυτή σημαντικό ρόλο κατά το σχεδιασμό.

4.1 MVC

Δουλεύοντας για κάποιο διάστημα με MVC frameworks, κυρίως το Robotlegs έγινε προφανές το πόσο βοηθάει η χρήση του MVC κατά το σχεδιασμό και την ανάπτυξη όποτε ήταν προφανές ότι θα πρέπει να ακολουθηθεί αυτή η σχεδιαστική αρχή. Φυσικά κατά τα πρώιμα στάδια της ανάπτυξης, λόγο της έλλειψης εξοικείωσης με την ανάπτυξη για την πλατφόρμα υπήρξαν αρκετά προβλήματα ώστε να επιτευχθεί ένας όσο το δυνατό σωστότερος διαχωρισμός των αρμοδιοτήτων.

Στην τρέχουσα μορφή της η εφαρμογή έχει το `MainDataModel` το οποίο διατηρεί όλα τα δεδομένα τα οποία λαμβάνει η εφαρμογή. Το model διαθέτει ένα αριθμό από πεδία για τα δεδομένα που διατηρεί, μαζί με τους κατάλληλους getters και setters. Ο ρόλος του model είναι να αποθηκεύει τα δεδομένα της εφαρμογής και να ενημερώνει με την χρήση events τα κατάλληλα views τα οποία στην περίπτωση μας αντιπροσωπεύονται από τα εκάστοτε fragments.

Όπως είδαμε στο προηγούμενο κεφάλαιο το μηνύματα είναι αντικείμενα από συγκεκριμένες classes τα οποία στέλνονται μέσω της μεθόδου `post()` του event bus. Η κλήση της μεθόδου post γίνεται πάντα στο τέλος του αντίστοιχου setter και αφού έχει αποθηκευτεί στην τοπική μεταβλητή η νέα τιμή. Όταν το μήνυμα σταλεί

κατά την αλλαγή κάποιας τιμή τότε όλοι οι subscribed handlers θα αναλάβουν να το διαχειριστούν και να ενημερώσουν το layout του fragment.

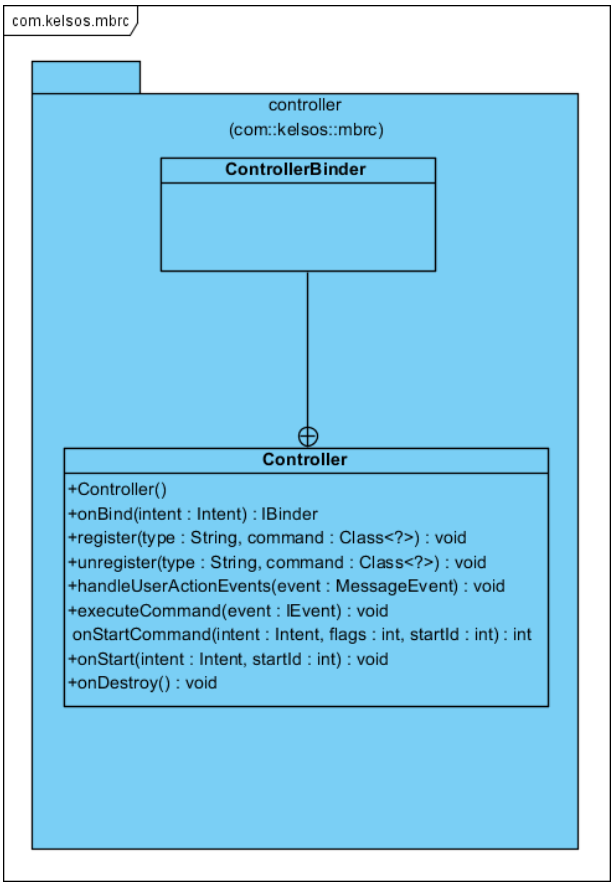
```
@Subscribe public void handleCoverEvent(final CoverAvailable cevent) {
  if (albumCover == null) return;
  if (cevent.getIsAvailable()) {
   albumCover.setImageBitmap(cevent.getCover());
  } else {
   albumCover.setImageResource(R.drawable.ic_image_no_cover);
  }
}
```

Στο παραπάνω παράδειγμα βλέπουμε τον handler για το `CoverAvailable` event το οποίο θα αποσταλεί όταν αλλάξει το εξώφυλλο που συσχετίζεται με το κομμάτι που παίζει. Κατά την εκτέλεση του ο handler θα αναλάβει να τροποποιήσει το image το οποίο παρουσιάζεται στο `ImageView albumCover`.

Επίσης το model διαθέτει και producer μεθόδους οι οποίες χρησιμεύουν ώστε να ενημερώνονται views από τα αποθηκευμένα δεδομένα κατά την εμφάνιση τους. Αυτή η προσέγγιση εξαλείφει την ανάγκη για αποθήκευση και ανάκτηση του state όπως θα έπρεπε να γίνει διαφορετικά για παράδειγμα στην περίπτωση του orientation change.

Αν και όπως αναφέρθηκε πιο πάνω τα fragments και τα activities του Android λειτουργούν ως views σε πολλές περιπτώσεις η τρέχουσα υλοποίηση αν και προσπαθεί να ακολουθήσει το MVC pattern μπορεί να έχει αρκετές διαφοροποιήσεις.

Το επόμενο τμήμα που αποτελεί την καρδιά της εφαρμογής είναι ο controller. Ο controller είναι μια Singleton class η οποία επεκτείνει την `Service.class` (`RoboService.class` λόγο της χρήσης του RoboGuice). Η επιλογή τις επέκτασης της service από τον controller έχει γίνει συνειδητά μιας και ο controller μπορεί κάλλιστα να εκτελείται στο background, κρατώντας μια socket σύνδεση ενεργεί για παράδειγμα αφού ο χρήστης έχει διώξει την εφαρμογή του από το προσκήνιο.

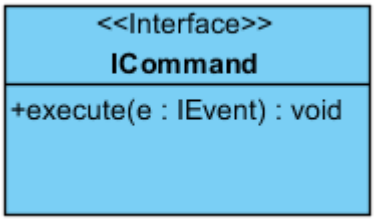


Εικόνα 14. Controller.class

Η εκκίνηση του controller γίνεται από την `onCreate()` του `RemoteApplication` με την κλήση της μεθόδου `startService()`. Αφότου κληθεί η μέθοδος έχουμε την εκτέλεση της callback μεθόδου `onStartCommand()` κατά την οποία ο controller κάνει register στο event bus ώστε να αρχίσει να ακούει για `MessageEvents` και στη συνέχει καλεί την στη συνέχεια καλείται η static μέθοδος `register()` της κλάσης `CommandRegistration`. Όπως θα δούμε η μέθοδος δέχεται τον controller ως παράμετρο και ο ρόλος της είναι να αρχικοποιήσει το configuration του controller.

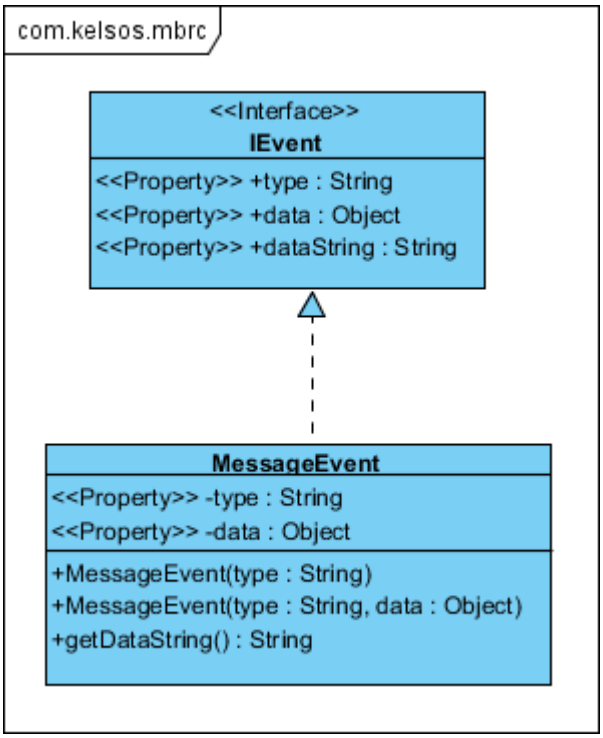
Ας σταθούμε προσωρινά στην υλοποίηση του controller. Αρχικά ο controller έχει μεθόδους για να κάνει register ή unregister κάποιο event. Το event αντιπροσωπεύεται από ένα String το οποίο πρέπει να είναι μοναδικό κάθε registration και ένα `Class` αντικείμενο (όλοι οι τύποι στην Java συμπεριλαμβανομένων των primitive και των πινάκων είναι συσχετισμένοι με ένα `Class` αντικείμενο).

Οι συνδυασμοί των String και Class αποθηκεύονται σε ένα `HashMap`, με κάθε κλειδί (string event type) να αποθηκεύετε μόνο μια φορά, δηλαδή την πρώτη που θα εισαχθεί στο `HashMap`.



Εικόνα 15. ICommand interface

Να σημειωθεί ότι αν και δεν γίνεται κάποιος έλεγχος ο στόχος είναι τα Class αντικείμενα τα οποία εισάγονται στο `HashMap` να υλοποιούν το `ICommand` interface. Όπως φαίνεται και στην Εικόνα 15 το `ICommand` περιέχει μόνο μια μέθοδο την μέθοδο `execute()` η οποία δέχεται ως παράμετρο ένα `IEvent`. Το `IEvent` είναι το interface από το οποίο κληρονομεί η `MessageEvent`.



Εικόνα 16. MessageEvent

Όπως αναφέρθηκε προηγουμένως ο controller κάνει register στο bus και αρχίζει να ακούει για `MessageEvents` τα οποία γίνονται post.

```
@Subscribe public void handleUserActionEvents(MessageEvent event) {...}
```

Αυτά τα μηνύματα τα χειρίζεται η μέθοδος `handleUserActionEvents()` η οποία στη συνέχεια περνάει το event στην `executeCommand()` μέθοδο του controller.

```
public void executeCommand(IEvent event) {
  Class<ICommand> commandClass = (Class<ICommand>)
this.commandMap.get(event.getType());
  if (commandClass == null) return;
  ICommand commandInstance;
  try {
```

```
   commandInstance = injector.getInstance(commandClass);
   if (commandInstance == null) return;
   commandInstance.execute(event);
  } catch (Exception ex) {
   if (BuildConfig.DEBUG) {
       Log.d("mbrc-log", "executing command for type: \t" + event.getType(),
ex);
       Log.d("mbrc-log", "command data: \t" + event.getData());
   }
  }
}
```

Η `executeCommand` προσπαθεί με τη χρήση της `getType()` να δει αν υπάρχει κάποιο Class αποθηκευμένο για το τρέχον event type (mapping). Αν δεν βρει κάτι και το αποτέλεσμα τις αναζήτησης στο `HashMap` είναι null τότε απλά επιστρέφει τερματίζοντας την εκτέλεση της μεθόδου. Στην περίπτωση που το βρει τότε ζητάει από τον injector να του φέρει ένα instance, κατά την κλήση αυτή ο injector θα εισάγει ότι αντικείμενα χρειάζονται (εφόσον έχουν σημανθεί σωστά με το `@Inject` annotation) και θα καλέσει την μέθοδο execute περνώντας της το event αντικείμενο.

Η ιδέα είναι ότι κάθε class που κληρονομεί την `ICommand` κάνει κάτι μικρό από πλευράς επεξεργασίας, για παράδειγμα η class `UpdateArtistSearchResults` διαβάζει ένα `ArrayNode` από τα δεδομένα που έχουν περάσει μαζί με το event και περνάει τα δεδομένα σε ένα `ArrayList` με `ArtistEntry` αντικείμενα. Να σημειωθεί ότι υπάρχει ένας αριθμός από model updating commands τα οποία στη συνέχεια αναλαμβάνουν να ενημερώσουν το model.

Ο τρέχον σχεδιασμός βοηθά στο να διατηρηθεί μικρός και γενικός ο controller, καθιστώντας την επαναχρησιμοποίηση του εφικτή σε μεγάλο βαθμό χωρίς να απαιτείται κάποια ιδιαίτερη τροποποίηση.

Καλό θα ήταν να αναφερθεί ότι στα πρώιμα στάδια της ανάπτυξης η σχεδίαση του controller ήταν αρκετά διαφορετική υπό την άποψη ότι ο διαχωρισμός της εκτέλεσης για κάθε διαφορετική ενέργεια γινόταν εσωτερικά στον controller με αποτέλεσμα, αυτός να έχει γνώση για κάθε δυνατή ενέργεια και για το πώς να τη διαχειριστεί. Υπό αυτή τη λογική μπορεί να θεωρηθεί ότι ήταν ένα god object και υπέφερε από υψηλή σύζευξη (high coupling) μιας έπρεπε να γνωρίζει όλα τα συνεργαζόμενα τμήμα. Αυτό καθιστούσε την επαναχρησιμοποίηση του controller ουσιαστικά αδύνατη.

Πηγή έμπνευσης για τον τρέχον σχεδιασμό υπήρξε το PureMVC framework για Java, και χρειάστηκε επανασχεδιασμός. Η τρέχουσα μορφή είναι αποτέλεσμα refactoring ώστε να βελτιωθεί ο κώδικας.

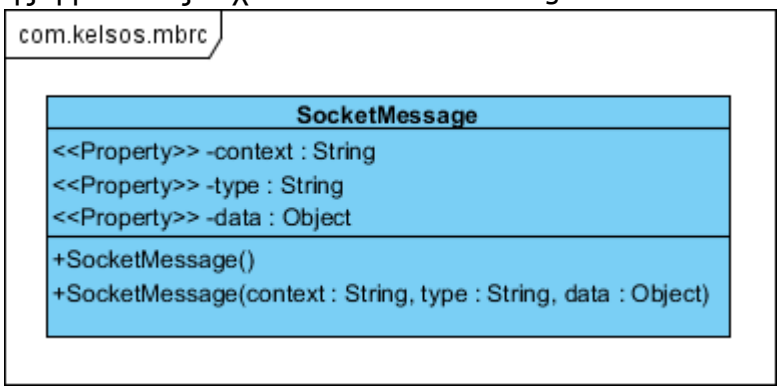
4.2 Client

Όπως αναφέρθηκε κατά την εισαγωγή η εφαρμογή λειτουργεί ως client για να πετύχει τη λειτουργικότητα της και συνδέετε με την χρήση ενός TCP socket στο plugin της εφαρμογής που λειτουργεί ως server. Η λειτουργικότητα αυτή έχει υλοποιηθεί με την χρήση της class `SocketService`.

Το TCP socket εκτελείται σε ξεχωριστό `Thread` στο παρασκήνιο και παρέχει αμφίδρομη επικοινωνία με το socket server ο οποίος είναι τμήμα του απομακρυσμένου plugin, λαμβάνοντας μηνύματα όταν και όποτε αυτά είναι

διαθέσιμα, για παράδειγμα όταν αλλάξει το κομμάτι που παίζει και στέλνοντας όποτε ο χρήστης εκτελέσει κάποια ενέργεια.

Η `SocketService` class διαχειρίζεται την έναρξη και τον τερματισμό της λειτουργίας του socket (αναλαμβάνει την διαχείριση του κύκλου ζωής του thread), και περιέχει μια μέθοδο `sendData()` για την αποστολή μηνυμάτων στο socket. Κατά την κλήση της η μέθοδος δέχεται ένα `SocketMessage`.



Εικόνα 17. SocketMessage

Η `sendData()` κάνει serialize το μήνυμα σε JSON και το στέλνει στο socket. Κάθε μήνυμα που αποστέλλεται ή λαμβάνεται μέσω του socket μετατρέπεται στην παρακάτω μορφή:

```
{"context":"","type", "data":""}
```

Το context είναι ένα string που αντιπροσωπεύει το μήνυμα και συνδέετε άμεσα με μια ενέργεια. Κάθε διαφορετικό context ζητά από το server μια διαφορετική ενέργεια και είναι ουσιαστικά οι εντολές[8] που υποστηρίζονται από το πρωτόκολλο που έχει υλοποιηθεί αποκλειστικά για την εφαρμογή. Για παράδειγμα αν στο πεδίο context βάλουμε την τιμή `playernext` τότε ζητάμε από το server την αναπαραγωγή του επόμενου κομματιού. Το type είναι επίσης ένα ακόμη string που αντιπροσωπεύει τον τύπο του μηνύματος, κυρίως για να γίνεται διαχωρισμός μεταξύ ερωτήσεων και απαντήσεων και απλών μηνυμάτων. Τέλος το τμήμα των data όπως φαίνεται και στην Εικόνα 17 είναι ένα object. Αυτό συμβαίνει γιατί ουσιαστικά ο τύπος των δεδομένων αλλάζει δυναμικά ανάλογα με την εντολή και το είδος της ενέργειας. Για παράδειγμα όταν ο χρήστης ζητήσει την τρέχουσα κατάσταση του Shuffle το data κομμάτι περιέχει μια Boolean τιμή, αν αλλάζει την ένταση της φωνής τότε αυτό που θα του επιστραφεί είναι ένα integer από το 0 μέχρι το 100, ενώ στην περίπτωση που ζητήσει την τρέχουσα λίστα αναπαραγωγή του τότε το data θα περιέχει ένα πίνακα με αντικείμενα που αντιπροσωπεύουν τα κομμάτια που βρίσκονται στη λίστα αναπαραγωγής.

Κατά την αποστολή του μηνύματος το κομμάτι περιλαμβάνει τμήμα των δεδομένων που συσχετίζονται μια την τρέχουσα ενέργεια. Για παράδειγμα κατά την αποστολή μιας εντολής για την αλλαγή της έντασης έχουμε τον αριθμό (integer) που αντιπροσωπεύει τη νέα ένταση, ή κατά την αναζήτηση ενός κομματιού μπορεί να περιέχει τον τίτλο.

[8] Οι δυνατές εντολές του πρωτοκόλλου βρίσκονται διαθέσιμες στην class `com.kelsos.mbrc.constants.Protocol`

Επιστρέφοντας στο σχεδιαστικό τμήμα του controller, υπάρχει τμήμα του configuration το οποίο συνδέει τα μηνύματα του πρωτοκόλλου άμεσα με υλοποιήσεις του `ICommand` interface. Το γεγονός αυτό μας επιτρέπει να αναθέσουμε ένα μικρό τμήμα κώδικα για κάθε ξεχωριστή εντολή το οποίο μπορεί για παράδειγμα να διαχειρίζεται διαβάζει τα δεδομένα να τα τροποποιεί αν είναι απαραίτητο και να ενημερώνει για παράδειγμα το `MainDataModel`, όπως γίνεται και στο παράδειγμα του `UpdateArtistSearchResults` που είδαμε πιο πάνω.

Πριν το πρωτόκολλο αποκτήσει την τρέχουσα μορφή του (χρήση JSON) το πρωτόκολλο χρησιμοποιούσε XML για την μεταφορά των δεδομένων. Η χρήση του xml αντικαταστάθηκε με τη χρήση JSON κυρίως λόγο του μικρότερου overhead που προσθέτει η σήμανση ιδιαίτερα στην περίπτωση πινάκων, γεγονός που πρέπει να λάβουμε υπόψιν, διότι αν και ο σχεδιασμός της εφαρμογής είναι κυρίως για χρήση μέσω Wi-Fi, αυτό δεν αποτρέπει την χρήση μέσω κάποιου mobile data pack, στα οποία το διαθέσιμο bandwidth είναι περιορισμένο και συνήθως κοστίζει.

Επίλογος

Στο κεφάλαιο αυτό είδαμε πληροφορίες για τον σχεδιασμό της εφαρμογής και για το πώς λειτουργεί ο controller, σε συνδυασμό με τα commands και το `MainDataModel` τα οποία αποτελούν τμήματα μιας υποτυπώδους MVC υλοποίησης η οποία εκμεταλλεύεται τα πλεονεκτήματα που της παρέχουν η χρήση του Dependency Injection framework σε συνδυασμό με το event bus.

Στη συνέχει έγινε μια σύντομη παρουσίαση του τρόπου λειτουργίας του client καθώς και του τρόπου λειτουργίας του πρωτοκόλλου της εφαρμογής συνδυασμό με το πώς η διαχείριση – επεξεργασία του πρωτοκόλλου συνδέετε με την τρέχουσα υλοποίηση του controller και των commands.

ΕΠΙΛΟΓΟΣ

Η ανάπτυξη της εφαρμογής αυτής υπήρξε σημαντική πηγή για την εκμάθηση της πλατφόρμας Android καθώς και χώρος εξερεύνησης τεχνολογιών από το χώρο της Java καθώς και τεχνικών και μεθόδων από το χώρο της μηχανικής λογισμικού οι οποίες βοήθησαν στον εμπλουτισμό γνώσεων και δεξιοτήτων. Το maven για παράδειγμα είναι ένα σημαντικό εργαλείο το οποίο χρησιμοποιείται από αρκετές εταιρίες στο χώρο της Java ιδιαίτερα για μεγάλα project.

Επίσης σημαντική επιρροή όσον αφορά τον σχεδιασμό και την υλοποίηση υπήρξαν το Dependency Injection pattern σε συνδυασμό με τη χρήση του event bus τα θα είναι σημαντικά εφόδια για την ανάπτυξη μελλοντικών εφαρμογών.

Αποτελεί σημαντικό στόχο για το μέλλον η ανάπτυξη αυτής της εφαρμογής να συνεχιστεί και μετά το πέρας αυτής της πτυχιακής εμπλουτίζοντας τη λειτουργικότητα της με επιπλέον χαρακτηριστικά, σε συνδυασμό με την περαιτέρω βελτίωση του κώδικα της.

ΣΥΜΠΕΡΑΣΜΑΤΑ – ΠΡΟΤΑΣΕΙΣ

Σημαντικό συμπέρασμα από την ανάπτυξη της εφαρμογής αποτελεί το γεγονός ότι από τη στιγμή που πρόκειται να γίνει διάθεση στο κοινό θα πρέπει να υπάρχει μια ισορροπία μεταξύ της προσπάθειας για καλύτερο σχεδιασμό και της ανάπτυξης καινούργιων χαρακτηριστικών. Αν γίνει επικέντρωση στο δεύτερο χωρίς να λαμβάνεται υπόψιν το πρώτο τότε μπορεί κάλλιστα σύντομα να υπάρχει μια εφαρμογή η οποία δεν είναι δυνατό να αναπτυχθεί περαιτέρω, από την άλλη αν επικεντρώνεις συνεχώς στην αναδόμηση και τη βελτίωση χωρίς να προσθέτεις καινούργια χαρακτηριστικά θα έχεις μια εφαρμογή η οποία δεν θα έχει να προσφέρει τίποτα καινούργιο στους χρήστες της. Επειδή ο χρόνος συνήθως είναι περιορισμένος η ισορροπία κρίνεται απαραίτητη.

Καλό θα ήταν να σημειωθεί ότι ο σχεδιασμός της εφαρμογής είναι κάθε άλλο παρά τέλειος, κάνοντας λοιπόν μια μικρή ανασκόπηση της ανάπτυξης το συμπέρασμα είναι ότι υπάρχουν αρκετά σημεία που χρίζουν βελτίωσης, καθώς και τροποποιήσεις που μπορούν να γίνουν για τη βελτίωση.

Σημαντικό επόμενο βήμα είναι η χρήση unit testing στο project, σκοπός είναι να χρησιμοποιηθεί το framework Mockito για την δημιουργία μεγάλου τμήματος των αντικειμένων (stub) τα οποία θα χρησιμοποιηθούν κατά τη διάρκεια του unit testing, καθώς και του Robolectic, το οποίο κάνει εφικτή τη δοκιμή τμημάτων κώδικα τα οποία εξαρτούνται από το Android SDK, παρέχοντας εικονικές υλοποιήσεις (shadow classes), γεγονός το οποίο επιτρέπει την εκτέλεση των unit test χωρίς να απαιτείται κάποια Android συσκευή εφόσον αυτά μπορούν να τρέξουν και στο κανονικό JVM.

Άλλο ένα σημαντικό πρόβλημα με την παρούσα υλοποίηση είναι η δημιουργία και χρήση πολλών μικρών προσωρινών αντικειμένων, είτε αυτά είναι υλοποιήσεις του `ICommand` interface, είτε αντικείμενα μηνυμάτων τα οποία μεταφέρονται μέσω του event bus. Αυτή η υλοποίηση θεωρείται κακή πρακτική από πλευράς απόδοσης γιατί η συνεχής κατανομή μνήμης (memory allocation) για την δημιουργία νέων αντικειμένων κοστίζει. Καθώς δημιουργούνται όλο και περισσότερα αντικείμενα θα πρέπει να γίνει κλήση στον Garbage Collector για την συλλογή αντικειμένων και την απελευθέρωση μνήμης, γεγονός που σε αρκετές περιπτώσεις μπορεί να έχει αντίκτυπο στην ομαλότητα της εφαρμογής.

Μια πιθανή λύση, η οποία χρησιμοποιείτε αρκετά από το Android είναι η χρήση του κατασκευαστικού σχεδιαστικού προτύπου (creational design pattern) Object Pool το οποίο διατηρεί ένα αρχικοποιημένο (initialized) σύνολο αντικειμένων έτοιμων για χρήση, αντί να τα δημιουργεί και να τα καταστρέφει κατά βούληση. Η χρήση του προτύπου μπορεί να βοηθήσει στην αποφυγή συχνών κλήσεων του Garbage Collector.

Σημαντικό επίσης είναι να γίνεται βελτίωση του κώδικα ώστε να γίνει καλύτερη εκμετάλλευση των δυνατοτήτων που του παρέχει το dependency injection framework.

Ακόμη θα ήταν επιθυμητό να γίνει καλύτερη χρήση των fragments κυρίως για την παροχή καλύτερης εμπειρίας στους χρήστες των tablet συσκευών. Κατά διάρκεια της ανάπτυξης υπήρξε μια προσπάθεια χρήσης διαφορετικού user

interface κυρίως στο landscape orientation όμως εγκαταλείφθηκε προσωρινά λόγο προβλημάτων και απόφασης να αφιερωθεί ο διαθέσιμος χρόνος στην προσθήκη λειτουργικότητας.

ΒΙΒΛΙΟΓΡΑΦΙΑ

AlertDialog.Builder | Android Developers. (n.d.). Ανάκτηση από Android Developers: https://developer.android.com/reference/android/app/AlertDialog.Builder.html

Android (operating system). (2013, #aug#). *Android (operating system)*. Ανάκτηση από http://en.wikipedia.org/w/index.php?title=Android_(operating_system)&oldid=568948571

Android Maven Plugin - android:apk. (n.d.). Ανάκτηση από Android Maven Plugin: http://maven-android-plugin-m2site.googlecode.com/svn/apk-mojo.html

Android Open Source Project. (n.d.). *Action Bar | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/guide/topics/ui/actionbar.html

Android Open Source Project. (n.d.). *Activity | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/reference/android/app/Activity.html#finish()

Android Open Source Project. (n.d.). *AlertDialog | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/reference/android/app/AlertDialog.html

Android Open Source Project. (n.d.). *Building and Running from the Command Line | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/tools/building/building-cmdline.html

Android Open Source Project. (n.d.). *Creating a Navigation Drawer | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/training/implementing-navigation/nav-drawer.html

Android Open Source Project. (n.d.). *Dashboards | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/about/dashboards/index.html

Android Open Source Project. (n.d.). *DialogFragment | Android Developers*. Ανάκτηση από https://developer.android.com/reference/android/app/DialogFragment.html

Android Open Source Project. (n.d.). *Fragments | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/guide/components/fragments.html

Android Open Source Project. (n.d.). *FragmentStatePagerAdapter | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/reference/android/support/v13/app/FragmentStatePagerAdapter.html

Android Open Source Project. (n.d.). *Handler | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/reference/android/os/Handler.html

Android Open Source Project. (n.d.). *Looper | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/reference/android/os/Looper.html

Android Open Source Project. (n.d.). *Navigation Drawer | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/design/patterns/navigation-drawer.html

Android Open Source Project. (n.d.). *NotificationCompat.Builder | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html

Android Open Source Project. (n.d.). *Notifications | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/guide/topics/ui/notifiers/notifications.html

Android Open Source Project. (n.d.). *Performance Tips | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/training/articles/perf-tips.html#object_creation

Android Open Source Project. (n.d.). *ProGuard | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/tools/help/proguard.html

Android Open Source Project. (n.d.). *Services | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/guide/components/services.html

Android Open Source Project. (n.d.). *Support Library | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/tools/support-library/index.html

Android Open Source Project. (n.d.). *Supporting Multiple Screens | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/guide/practices/screens_support.html

Android Open Source Project. (n.d.). *Toast | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/reference/android/widget/Toast.html

Android Open Source Project. (n.d.). *Using ViewPager for Screen Slides | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/training/animation/screen-slide.html

Android Open Source Project. (n.d.). *ViewPager | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/reference/android/support/v4/view/ViewPager.html

Android Open Source Project. (n.d.). *zipalign | Android Developers*. Ανάκτηση από Android Developers: https://developer.android.com/tools/help/zipalign.html

Apache Maven. (n.d.). Ανάκτηση από Wikipedia, the free encyclopedia: https://en.wikipedia.org/w/index.php?title=Apache_Maven

Apache Software Foundation. (2013, October 04). *Maven - What is Maven?* Ανάκτηση από Apache Maven Project: https://maven.apache.org/what-is-maven.html

Apache Software Foundation. (2013, April 03). *Maven Compiler plugin - Introduction*. Ανάκτηση από Apache Maven Project: https://maven.apache.org/plugins/maven-compiler-plugin/

Apache Software Foundation. (n.d.). *Maven - Introduction to the Dependency Mechanism*. Ανάκτηση από Apache Maven Project:

https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html

Apache Software Foundation. (n.d.). *Maven - Introduction to the Standard Directory Layout*. Ανάκτηση από Apache Maven Project: https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

Apache Software Foundation. (n.d.). *Maven - POM Reference*. Ανάκτηση από Apache Maven Project: https://maven.apache.org/pom.html

Binder | Guice. (n.d.). Ανάκτηση από http://google-guice.googlecode.com/git/javadoc/com/google/inject/Binder.html

Broyer, T. (2013, September 26). *Maven is broken by design*. Retrieved from tbroyer's pages: http://blog.ltgt.net/maven-is-broken-by-design/

Crawford, D. (2013, July 09). *Why mobile web apps are slow*. Ανάκτηση από sealed abstract: http://sealedabstract.com/rants/why-mobile-web-apps-are-slow/

Crouton. (n.d.). Ανάκτηση από GitHub: https://github.com/keyboardsurfer/Crouton

Dependency injection. (n.d.). Ανάκτηση από Wikipedia, the free encyclopedia: https://en.wikipedia.org/w/index.php?title=Dependency_injection

DragSortController | drag-sort-listview. (n.d.). Ανάκτηση από GitHub: http://bauerca.github.io/drag-sort-listview/reference/com/mobeta/android/dslv/DragSortController.html

drag-sort-listview. (n.d.). Ανάκτηση από GitHub: https://github.com/bauerca/drag-sort-listview

FasterXML LLC. (n.d.). *JacksonDataBinding - FasterXML Wiki*. Ανάκτηση από FasterXML Wiki: http://wiki.fasterxml.com/JacksonDataBinding

FasterXML LLC. (n.d.). *JacksonTreeModel - FasterXML Wiki*. Ανάκτηση από FasterXML Wiki: http://wiki.fasterxml.com/JacksonTreeModel

FasterXML LLC. (n.d.). *JsonNode (jackson-databind 2.1.0 API)*. Ανάκτηση από Jackson Documentation: http://fasterxml.github.io/jackson-databind/javadoc/2.1.0/

Fowler, M. (2004, January 23). *Inversion of Control Containers and the Dependency Injection pattern*. Ανάκτηση από Martin Fowler: http://martinfowler.com/articles/injection.html

Gartner, Inc. (n.d.). *Gartner Says Smartphone Sales Grew 46.5 Percent in Second Quarter of 2013 and Exceeded Feature Phone Sales for First Time*. Ανάκτηση από http://www.gartner.com/newsroom/id/2573415

GettingStarted - maven-android-plugin - Get started using Android Maven Plugin - (renamed to android-maven-plugin) Easy to use Maven plugin for Android - Google Project Hosting. (n.d.). Ανάκτηση από Google Project Hosting: http://code.google.com/p/maven-android-plugin/wiki/GettingStarted

God object. (n.d.). Ανάκτηση από Wikipedia, the free encyclopedia: https://en.wikipedia.org/w/index.php?title=God_object&oldid=557365634

Google Guice. (n.d.). Ανάκτηση από Wikipedia, the free encyclopedia: https://en.wikipedia.org/w/index.php?title=Google_Guice

Gradleware, Inc. (n.d.). *Gradle: The New Android Build System | Gradleware*. Ανάκτηση από Gradleware: http://www.gradleware.com/resources/tech/android

Gueye, L. (2012, May 27). *Android from scratch, part 2: use android-maven-plugin*. Ανάκτηση από Diving deep into JEE: http://deepintojee.wordpress.com/2012/05/27/android-from-scratch-part-2-use-android-maven-plugin/

Home · roboguice/roboguice Wiki. (n.d.). Ανάκτηση από Github: https://github.com/roboguice/roboguice/wiki

IntelliJ IDEA - Wikipedia, the free encyclopedia. (n.d.). Ανάκτηση από Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/IntelliJ_IDEA

Introduction to Google Guice. (2007, August 04). Ανάκτηση από JavaBeat: http://www.javabeat.net/2007/08/introduction-to-google-guice/#

Jenkov, J. (n.d.). *Java Reflection: Classes*. Ανάκτηση από Efficiency Matters: http://tutorials.jenkov.com/java-reflection/classes.html

Jenkov, J. (n.d.). *What is Dependency Injection?* Ανάκτηση από Efficiency Matters: http://tutorials.jenkov.com/dependency-injection/index.html

Jetbrains. (n.d.). *IntelliJ IDEA :: Java refactoring plus sophisticated code refactoring for {JSP}, {XML}, {CSS}, {HTML}, {JavaScript}*. Ανάκτηση από https://www.jetbrains.com/idea/features/refactoring.html

Jetbrains. (n.d.). *IntelliJ IDEA :: On-the-fly Code Analysis*. Ανάκτηση από https://www.jetbrains.com/idea/features/code_analysis.html

Jetbrains. (n.d.). *IntelliJ IDEA :: Smart Code Completion*. Ανάκτηση από https://www.jetbrains.com/idea/features/code_completion.html

Lesiecki, N. (2008, December 09). *Dependency injection with Guice*. Ανάκτηση από developerWorks: http://www.ibm.com/developerworks/java/library/j-guice/index.html

Martin, C. R. (2009). *Clean code: a handbook of agile software craftsmanship.* Upper Saddle River, {NJ}: Prentice Hall.

Maven Snapshots. (n.d.). Ανάκτηση από tutorialspoint: http://www.tutorialspoint.com/maven/maven_snapshots.htm

Maven vs Ant for automatic builds in Android applications - Stack Overflow. (n.d.). Ανάκτηση από Stack Overflow: http://stackoverflow.com/questions/6243716/maven-vs-ant-for-automatic-builds-in-android-applications

Maven: The Complete Reference - Sonatype.com. (n.d.). Ανάκτηση από Sonatype.com: http://books.sonatype.com/mvnref-book/reference/android-dev-sect-goals-internal.html

maven-android-sdk-deployer. (n.d.). Ανάκτηση από GitHub: https://github.com/mosabua/maven-android-sdk-deployer

Mobile operating system. (n.d.). Ανάκτηση από Wikipedia, the free encyclopedia: https://en.wikipedia.org/w/index.php?title=Mobile_operating_system

Modules | Guice. (n.d.). Ανάκτηση από Guice Documentation: https://google-guice.googlecode.com/git/javadoc/com/google/inject/util/Modules.html#override(com.google.inject.Module...)

Moser, M. (2012, October 22). *ZipalignAPKBuiltByMAven*. Ανάκτηση από https://code.google.com/p/maven-android-plugin/wiki/ZipalignAPKBuiltByMAven: https://code.google.com/p/maven-android-plugin/wiki/ZipalignAPKBuiltByMAven

New Build System - Android Tools Project Site. (n.d.). Ανάκτηση από Android Tools Project Site: http://tools.android.com/tech-docs/new-build-system

Object pool pattern - Wikipedia, the free encyclopedia. (n.d.). Ανάκτηση από Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Object_pool_pattern

Post, M. (2012, January 15). *Serious unit testing on Android*. Ανάκτηση από EclipseSource: http://eclipsesource.com/blogs/2012/06/15/serious-unit-testing-on-android/

roboguice. (n.d.). Ανάκτηση από GitHub: https://github.com/roboguice/roboguice

Schwarz, N., Lungu, M., & Nierstrasz, O. (2012). Seuss: Decoupling responsibilities from static methods for fine-grained configurability. *The Journal of Object Technology, 11*(1), 3:1. Ανάκτηση από http://www.jot.fm/contents/issue_2012_04/article3.html

Square, Inc. (n.d.). *Otto*. Ανάκτηση από GitHub: http://square.github.io/otto/

Test stub - Wikipedia, the free encyclopedia. (n.d.). Ανάκτηση από Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Test_stub

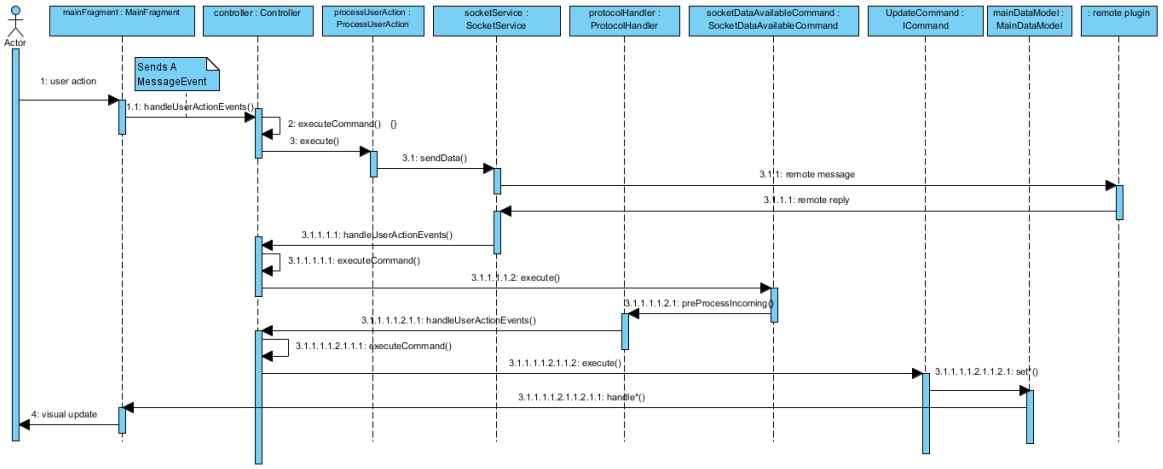
Vanbrabant, R. (2008). *Google Guice agile lightweight dependency injection framework.* Berkeley, {CA;} New York: Apress ; Distributed to the Book trade in the United States by Springer-Verlag.

Wharton, J. (n.d.). *ActionBarSherlock - Usage*. Ανάκτηση από ActionBarSherlock: http://actionbarsherlock.com/usage.html

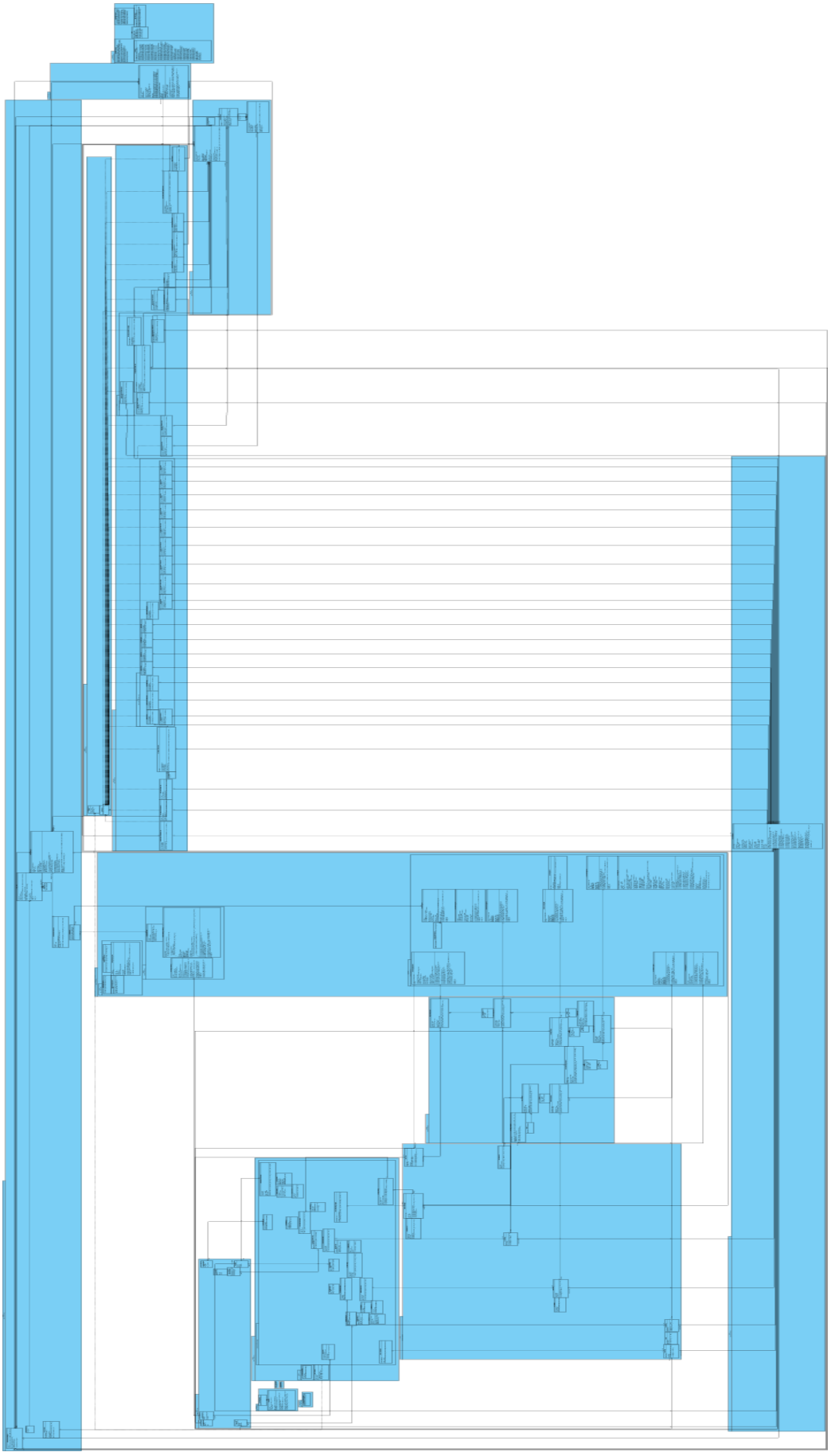
Wharton, J. (n.d.). *JakeWharton/ActionBarSherlock*. Ανάκτηση από GitHub: https://github.com/JakeWharton/ActionBarSherlock

Wharton, J. (n.d.). *ViewPagerIndicator*. Ανάκτηση από http://viewpagerindicator.com/

ΠΑΡΑΡΤΗΜΑ



Εικόνα 18. Sequence Diagram of basic interaction



Εικόνα 19. com.kelsos.mbrc class diagram

ΟΔΗΓΟΣ ΧΡΗΣΗΣ ΛΟΓΙΣΜΙΚΟΥ

Πριν τη χρήση του IDEA για το build και το deploy της εφαρμογής απαιτείται μια προεργασία η οποία περιλαμβάνει την εγκατάσταση του Oracle JDK (έκδοση 7 τη στιγμή που γράφονται αυτές οι γραμμές), του Android SDK για το Android 4.3 (API 18) (ή όποιας έκδοσης είναι η τελευταία διαθέσιμη) σε συνδυασμό με το Android Support Repository από τα Extras και το Google Repository με τη χρήση του Android SDK Manager. Απαιτείται επίσης να είναι εγκατεστημένο το git καθώς και η κατάλληλη έκδοση του Apache Maven (καλό θα ήταν να γίνει ένας έλεγχος των εκδόσεων που υποστηρίζει το android maven plugin).

Πριν προχωρήσουμε στο IDEA αρχικά θα πρέπει να εγκαταστήσουμε τα κατάλληλα artifacts στο τοπικό repository, αυτό θα επιτευχθεί με το εργαλείο Maven Android SDK Deployer που αναφέρθηκε στο πρώτο κεφάλαιο.

Σε κάποιο προσωρινό φάκελο κάνουμε ένα clone του repository του:

```
git clone https://github.com/mosabua/maven-android-sdk-deployer.git
cd maven-android-sdk-deployer
mvn install -P4.3
```

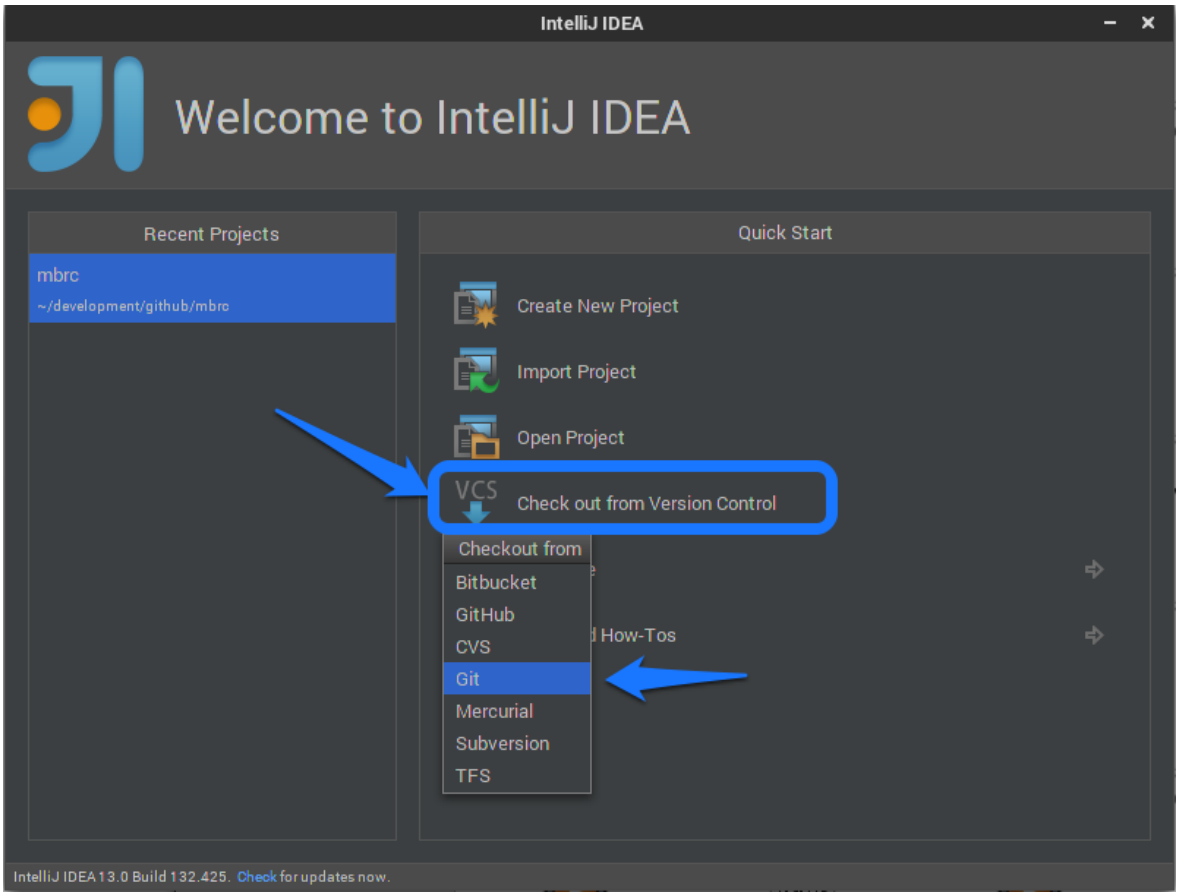
Στη συνέχεια μεταβαίνουμε στο φάκελο που έχει δημιουργηθεί και εκτελούμε την εντολή `mvn install` ώστε να εγκαταστήσουμε τα απαραίτητα artifacts στο τοπικό repository.

Επίσης είναι απαραίτητο να κατεβάσουμε το DragSortListView από το Github και να γίνει build από τα sources μιας και δεν είναι διαθέσιμο από το Maven Central. Με αυτό τον τρόπο το κάνουμε διαθέσιμο στο τοπικό repository

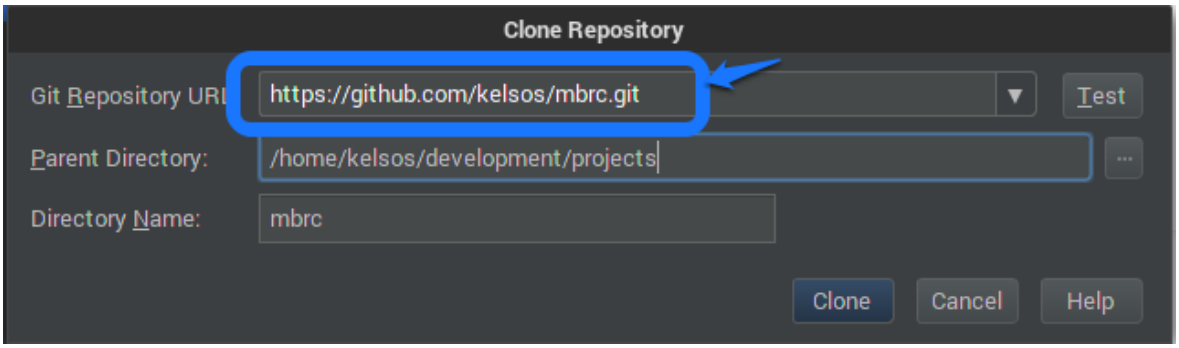
```
git clone https://github.com/kelsos/drag-sort-listview.git
cd drag-sort-listview
mvn clean install
```

Το παραπάνω fork περιέχει κάποιες μικροδιορθώσεις ώστε η βιβλιοθήκη να μπορεί να κάνει build με τις νεότερες εκδόσεις του Android SDK.

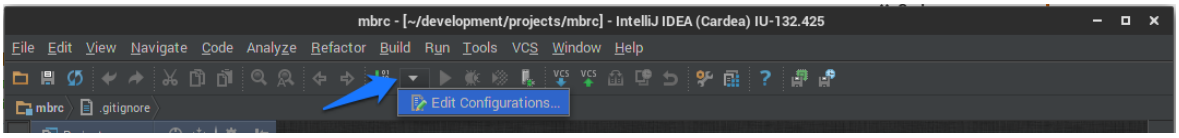
Αφού πραγματοποιηθούν οι ενέργειες που αναφέρθηκαν πιο πάνω μπορούμε να ξεκινήσουμε το IDEA. Ξεκινώντας το IDEA μπορούμε να δούμε την οθόνη καλωσορίσματος (welcome screen) από εκεί μπορούμε να ανοίξουμε κάποια από τα πρόσφατα Project, να δημιουργήσουμε νέα, να ανοίξουμε κάποια από τα υπάρχοντα ή να τροποποιήσουμε τις ρυθμίσεις η να κάνουμε checkout κώδικα από κάποιο απομακρυσμένο repository.



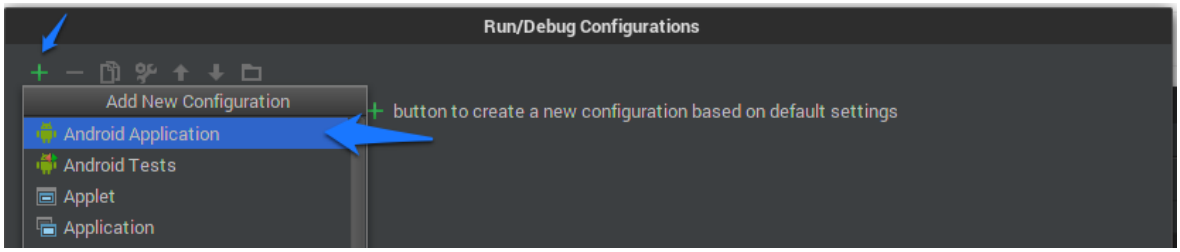
Στην περίπτωση μας θα επιλέξουμε να κάνουμε checkout των κώδικα από το git repository του project το οποίο φιλοξενείται στο Github (https://github.com/kelsos/mbrc.git).



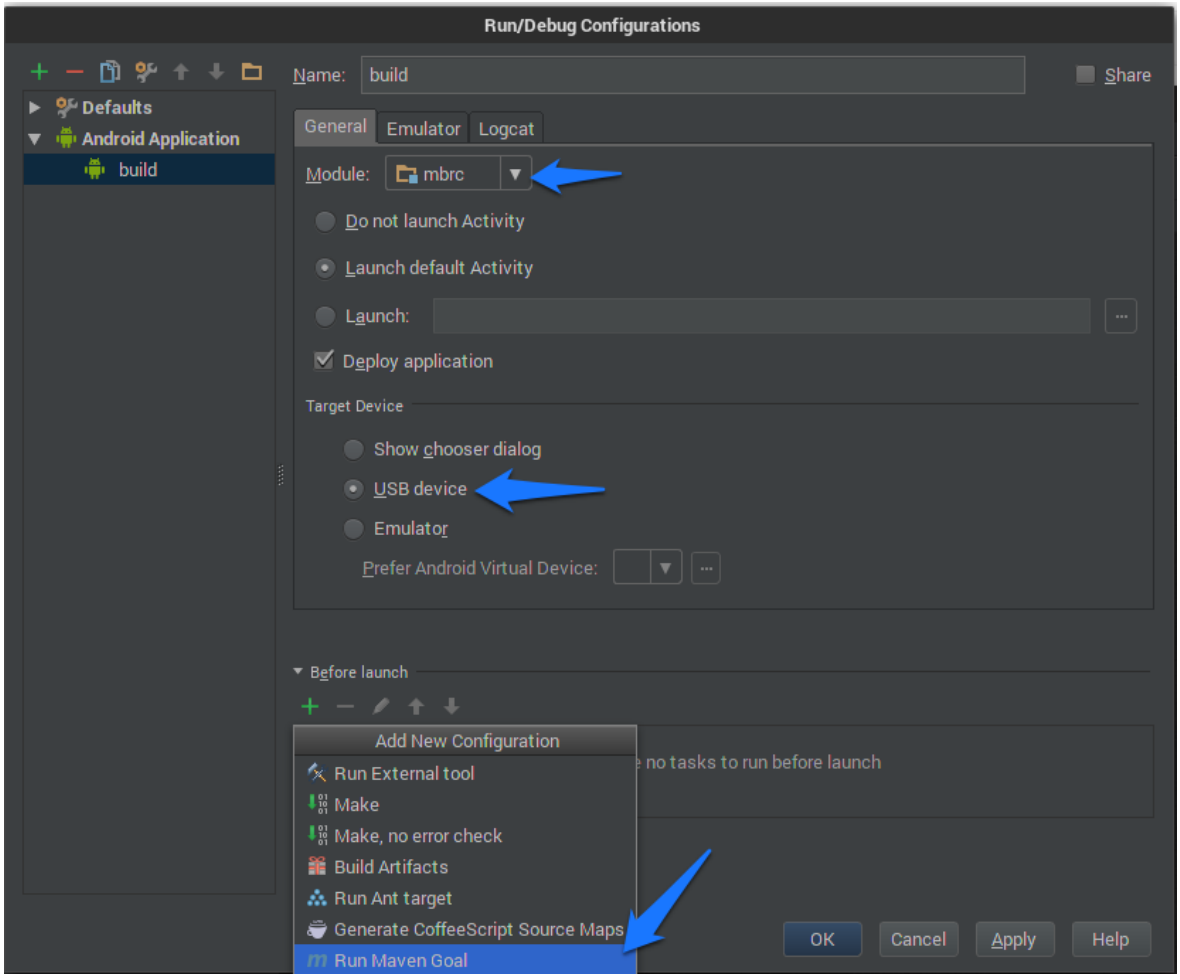
Εφόσον το IDEA κάνει checkout τον κώδικα από το Github repository θα μας ζητηθεί να ανοίξουμε το project. Ένα από τα πρώτα βήματα στο νέο project είναι να ορίσουμε το configuration του.



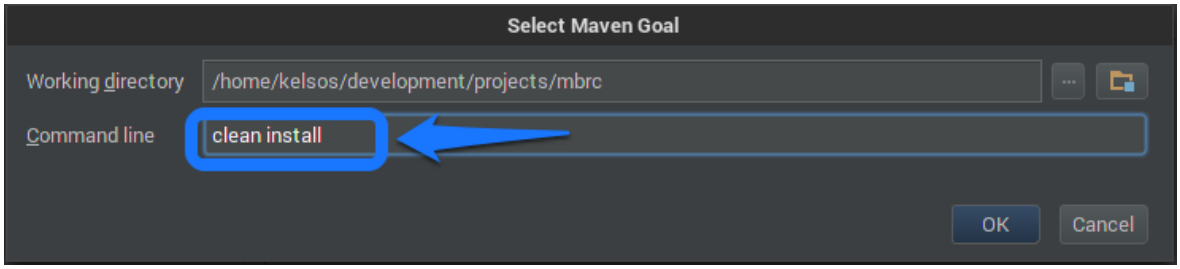
Για να το πετύχουμε αυτό όταν ανοίξει το παράθυρο Run/Debug Configurations επιλέγουμε να προσθέσουμε ένα νέο configuration για Android Application.



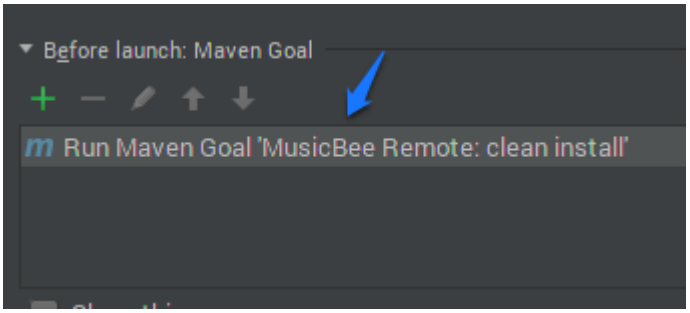
Στο νέο configuration που θα δημιουργηθεί επιλέγουμε ως module το mbrc που είναι αυτό που περιέχει το project μας. Από εδώ μπορούμε να καθορίσουμε τι ακριβώς θα γίνεται κάθε φορά που ο χρήστης πατά το build η debug κουμπί του IDE. Μπορούμε να διαλέξουμε για παράδειγμα αν θα γίνεται deploy η εφαρμογή σε κάποια συσκευή που είναι συνδεδεμένη μέσω USB η αν θα πρέπει να εκτελείται ο emulator.



Ένα κομμάτι που μας ενδιαφέρει αρκετά είναι το τι ακριβώς πρέπει να συμβεί πριν την εκτέλεση της εφαρμογής. Αυτό μπορεί να οριστεί στο πεδίο `Before launch`. Εκεί έχουμε για παράδειγμα την δυνατότητα να ορίσουμε την εκτέλεση κάποιων goals του maven.

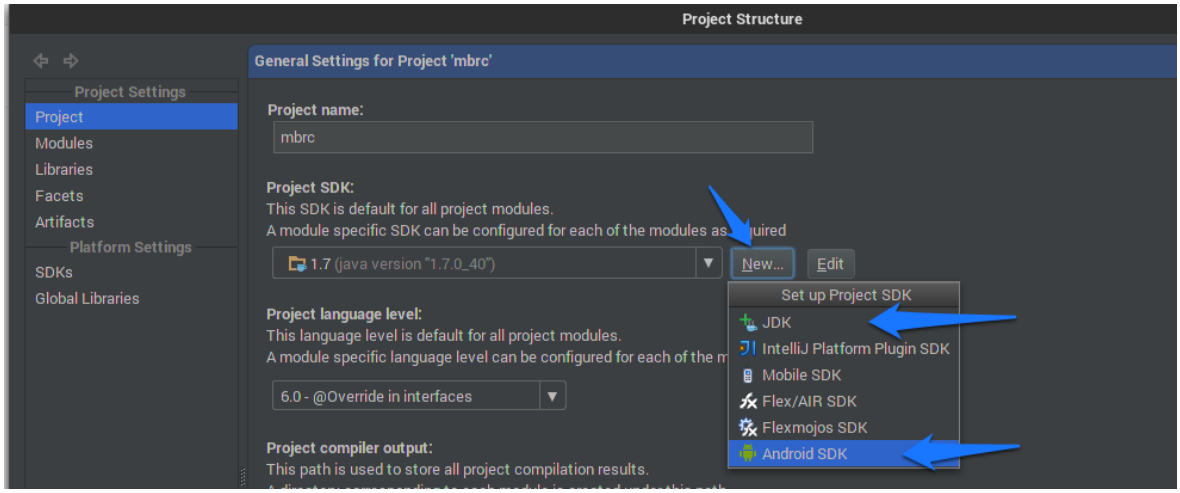


Στην περίπτωση μας, μας ενδιαφέρει η εκτέλεση των goal clean και install πριν από κάθε εκτέλεση ώστε να καθαρίζονται τα αρχεία του προηγούμενου build και να ξαναγίνεται build από την αρχή το project.

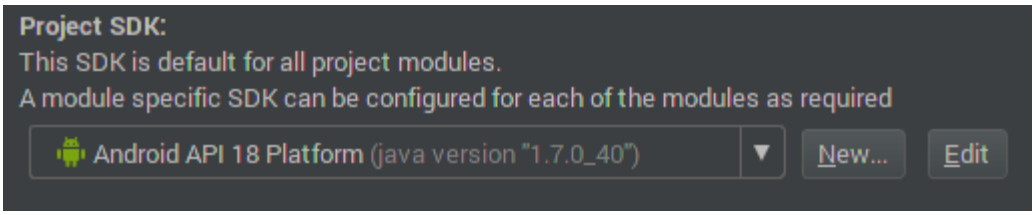


Να σημειωθεί ότι μπορούμε να δημιουργήσουμε όσα configuration θεωρούμε απαραίτητο, με διαφορετικές ρυθμίσεις το καθένα, για παράδειγμα μπορούμε να κάνουμε ένα το οποίο θα εκτελεί κάθε φορά την ίδια εφαρμογής χωρίς να δημιουργεί καινούργιο build κάθε φορά, το οποίο μπορεί να είναι χρήσιμο στην περίπτωση που τεστάρουμε κάτι αλλά δεν έχουμε πραγματοποιήσει κάποια αλλαγή στο source code.

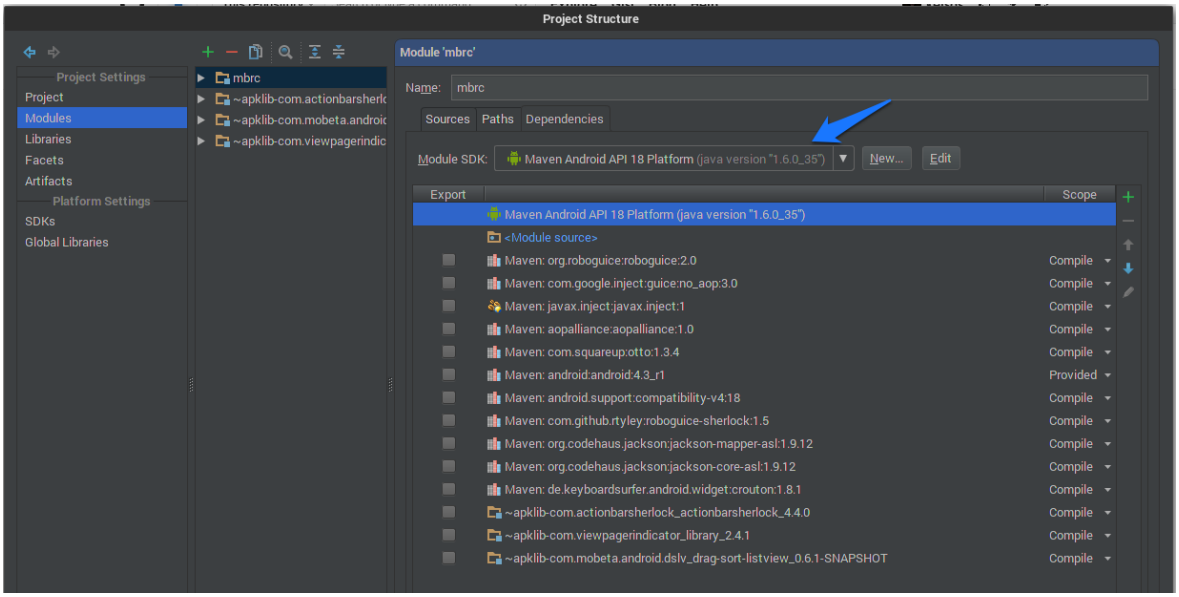
Στο επόμενο βήμα χρειάζεται να πάμε στο **File – Project Structure** και να ρυθμίσουμε το SDK. Αρχικά επιλέγουμε New και στην συνέχεια JDK ώστε να ενημερώσουμε το IDE για την τοποθεσία στην οποία βρίσκεται εγκατεστημένο το JDK. Έπειτα επιλέγουμε το New και στη συνέχεια το Android SDK ώστε να ενημερώσουμε το IDE για την τοποθεσία του Android SDK.



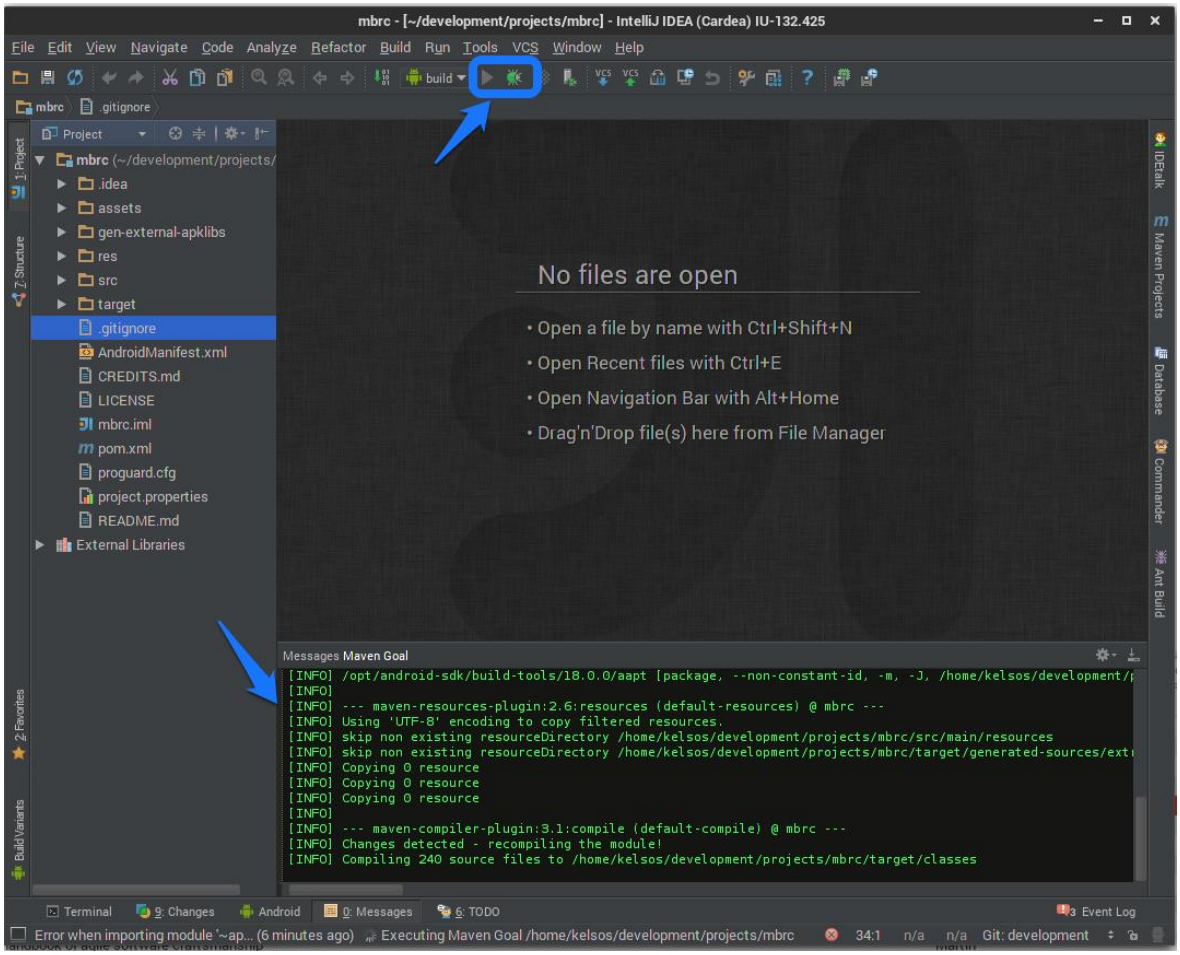
Μετά από μία επιτυχημένη ρύθμιση θα πρέπει να είναι ορατό στο Project SDK πεδίο κάτι σαν το παρακάτω:



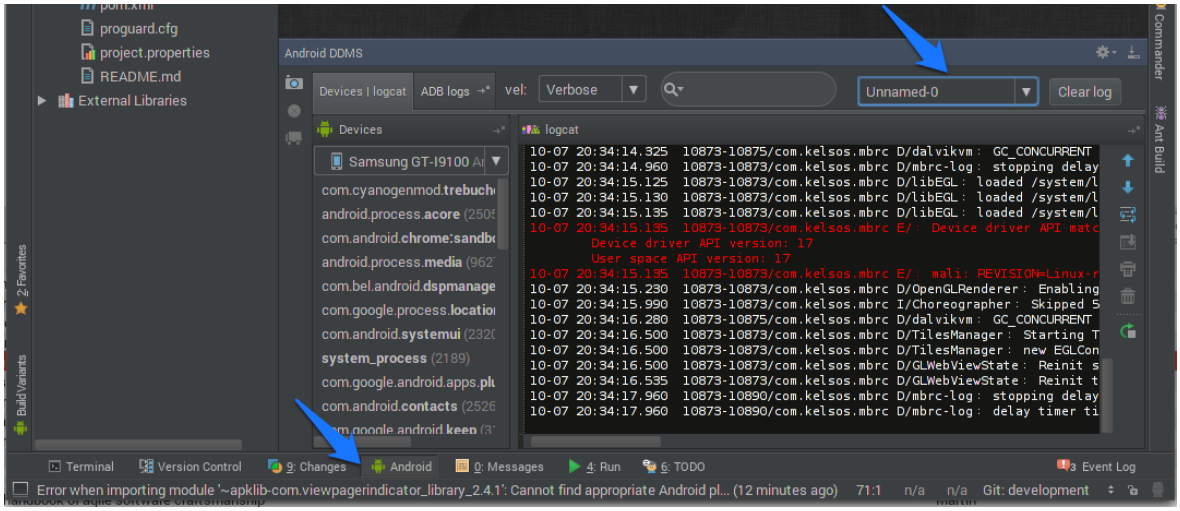
Υπάρχει η πιθανότητα αυτό το βήμα να μην απαιτείται λόγο της χρήσης του Maven και οι απαραίτητες ρυθμίσεις για το Project SDK να είναι ήδη διαθέσιμες από το drop down menu.



Σε κάθε περίπτωση καλό θα ήταν να επιβεβαιωθεί ότι το κατάλληλο module SDK έχει επιλεχθεί. Επίσης καλό θα ήταν να γίνει ένας έλεγχος στα διάφορα apklibs (modules) ώστε να επιβεβαιωθεί ότι έχουν ρυθμιστεί να χρησιμοποιούν το ίδιο SDK με το υπόλοιπο project (επιλογή Project SDK).

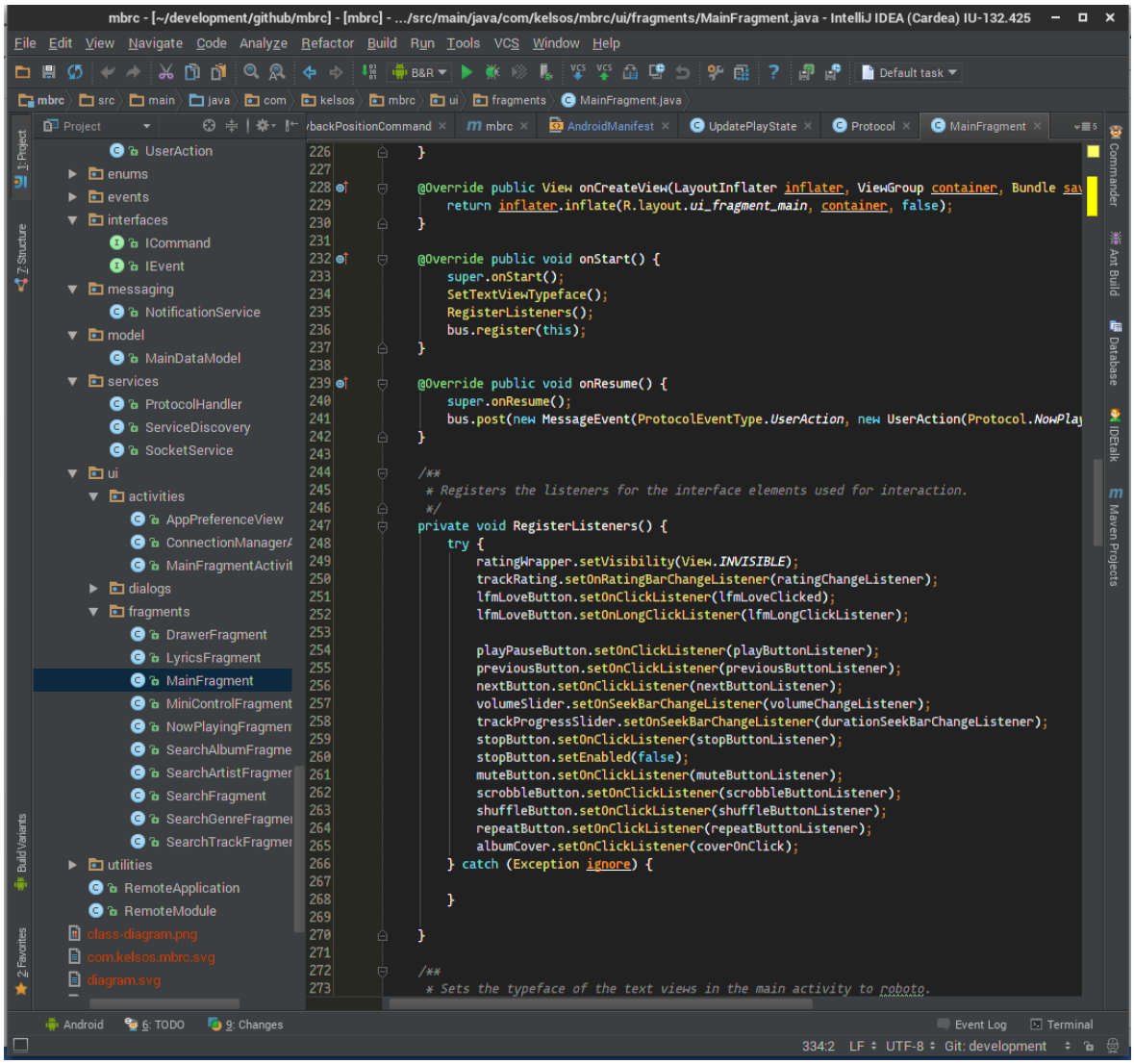


Τώρα πατώντας το κουμπί Run ή αντίστοιχα το Debug θα γίνει αρχικά η εκτέλεση των maven goals clean και install, στη συνέχεια η εφαρμογή θα εγκατασταθεί στην συνδεδεμένη με USB Android συσκευή και στη συνέχει θα γίνει η εκτέλεση αυτής.



Επιλέγοντας το Android στο κάτω μέρος της οθόνης ανοίγουμε το παράθυρο το οποίο ενσωματώνει το Android DDMS στο οποίο είναι ορατό το logcat, οι συνδεμένες συσκευές καθώς και μια λίστα με εφαρμογές οι οποίες εκτελούνται στην επιλεγμένη συσκευή. Το Unnamed-0 είναι ένα φίλτρο για το

logcat, μπορούμε να ορίσουμε τέτοια φίλτρα ώστε να αποφεύγουμε το «θόρυβο» που δημιουργείται από άλλες εφαρμογές.



Στο κεντρικό παράθυρο του IDEA βλέπουμε στα αριστερά τον project explorer, ο explorer χρωματίζει τα αρχεία ανάλογα με την κατάσταση τους στο version control system. Για παράδειγμα τα κόκκινα αρχεία είναι αρχεία τα οποία δεν βρίσκονται στο vcs. Στα δεξιά βλέπουμε τον editor ενώ στα γύρω παράθυρα βλέπουμε κουμπιά που ανοίγουν παράθυρα για επιπλέον λειτουργικότητα. Κάτω δεξιά μπορούμε να δούμε πληροφορίες για τη γραμμή και τη στήλη στην οποία βρίσκεται ο cursor, το line ending που χρησιμοποιείται στα αρχεία, καθώς και το character encoding, ενώ δίπλα μας δείχνει πληροφορία για το vcs (Git) καθώς και το τρέχον branch. Κάποιες βασικές λειτουργίες παρέχονται από το context menu του editor ενώ άλλες είναι διαθέσιμες μέσω του menu.