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

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Γραφική Γλώσσα Αιτημάτων στηριζόμενο στην NoSQL τεχνολογία για την πλατφόρμα Android

Των φοιτητων

Χαριτάκη Γεωργίου

Σεϊτανίδη Ηλία-Νεκταρίου

Αρ. Μητρώου: 093456(Χαριτάκης)-113793(Σεϊτανίδης)

Επιβλέπων καθηγητής

κ. Κεραμόπουλος Ευκλείδης

Θεσσαλονίκη 2016

ΠΡΟΛΟΓΟΣ

Η παρούσα πτυχιακή εργασία πραγματοποιήθηκε στο Αλεξάνδρειο Τεχνολογικό Ίδρυμα Θεσσαλονίκης, στο τμήμα Μηχανικών Πληροφορικής της Σχολής Τεχνολογικών Εφαρμογών. Η εργασία έχει ως σκοπό την ανάπτυξη, σε Mobile περιβάλλον, επιλέχθηκε το λειτουργικό σύστημα Android [1], μίας εφαρμογής η οποία είναι υπεύθυνη για την πραγματοποίηση ερωταπαντήσεων με τη χρήση Γραφικού Περιβάλλοντος (Graphical User Interface). Ο τύπος της Βάσης Δεδομένων για τον οποίο έχει σχεδιαστεί η συγκεκριμένη εφαρμογή είναι η db4o[2], μια NoSQL[3] Βάση Δεδομένων.

ΠΕΡΙΛΗΨΗ

Στόχος της πτυχιακής εργασίας είναι η δημιουργία ενός εύχρηστου Γραφικού Περιβάλλοντος σύνταξης αιτημάτων, για την πλατφόρμα Android, το οποίο θα στηθεί πάνω από κατάλληλο μοντέλο Βάσης Δεδομένων για υποστήριξη NoSQL τεχνολογίας, και πιο συγκεκριμένα για τις Βάσεις Δεδομένων Db4o. Η σχεδίαση του γραφικού περιβάλλοντος θα απευθύνεται σε χρήστες μη-σχετικούς με την τεχνολογία και τις ιδιαιτερότητάς του μοντέλου Βάσεων Δεδομένων που θα χρησιμοποιηθεί και της NoSQL τεχνολογίας. Ακόμα, για την δοκιμή της εφαρμογής καθ' ολη τη διάρκεια της ανάπτυξής δημιουργήσαμε ένα πρόγραμμα Εξυπηρετητή (Server) για τη Βάση Δεδομένων db4o.

Στην εργασία που ακολουθεί θα γίνει αναφορά στις βασικές τεχνολογίες που χρησιμοποιήθηκαν. Στη συνέχεια θα πραγματοποιηθεί εκτενής αναφορά τόσο στο πρόγραμμα Εξυπηρετητή όσο και στο Γραφικό Περιβάλλον και στον κώδικα της εφαρμογής.

ABSTRACT

The aim of the thesis is to create User-Friendly Graphical Interface query builder for the NoSQL Database System Db4o and devices powered by Android Operating System. Users with no or little knowledge of the NoSQL technology will be able to build a query by using this application. In the course of the application development we built an application that acts as a db4o server.

In the labor that follows there will be an extensive reference to both the server side and the client side applications.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα θέλαμε να ευχαριστήσουμε τον Επίκουρο Καθηγητή του Τμήματος Μηχανικών Πληροφορικής του Αλεξάνδρειου Εκπαιδευτικού Ιδρύματος Θεσσαλονίκης κύριο Ευκλείδη Κεραμόπουλο που μας έδωσε την ευκαιρία να ασχοληθούμε με αυτή τη θεματολογία, την ανεκτίμητη βοήθεια και καθοδήγηση που μας προσέφερε τόσο κατά τη διάρκεια εκπόνησης της πτυχιακής όσο και κατά τη διάρκεια της Φοιτητικής μας εκπαίδευσης μέσα από τα μαθήματα που διδάσκει.

Περιεχόμενα

Ευρετήριο Εικόνων

ΕΙΣΑΓΩΓΗ

Στόχος της πτυχιακής είναι η δημιουργία ενός εύχρηστου γραφικού περιβάλλοντος σύνταξης αιτημάτων για χρήστες μη-σχετικούς με την τεχνολογία και τις ιδιαιτερότητας του μοντέλου Βάσεων Δεδομένων που θα χρησιμοποιηθεί και της NoSQL τεχνολογίας, για συσκευές με λειτουργικό σύστημα Android [1]. Μετά την ολοκλήρωση της πτυχιακής, η εφαρμογή θα δίνει τη δυνατότητα σε χρήστη με μηδενικές γνώσεις πληροφορικής να είναι σε θέση να υποβάλλει ερωτήματα σε μια NoSQL (Not Only SQL) [3] βάση δεδομένων. Η βάση που θα χρησιμοποιηθεί στην πτυχιακή είναι η db4o [2]. Στα ακόλουθα κεφάλαια θα αναλυθούν οι μεθοδολογίες και οι περιορισμοί που θέτουν τόσο το λειτουργικό σύστημα Android (μέγεθος οθόνης, πόροι συστήματος, περιορισμοί στις δυνατότητες που προσφέρει το λειτουργικό σύστημα), όσο και η βάση δεδομένων db4o (μη εγγενής υποστήριξη από το λειτουργικό σύστημα Android, περιορισμοί στην δημιουργία και εμφάνιση των ερωτημάτων).

Για την εκπόνηση της πτυχιακής/έρευνας χρησιμοποιήθηκε το ολοκληρωμένο περιβάλλον ανάπτυξης Android Studio [4], το οποίο βασίζεται στο IntelliJ IDEA [5] και αποτελεί το βασικό IDE για την ανάπτυξη εφαρμογών Android. Οι δοκιμές της εφαρμογής έγιναν με συσκευές Lg Nexus 4 με την τελευταία έκδοση του λογισμικού Android.

ΚΕΦΑΛΑΙΟ 1

Τεχνολογίες και χαρακτηριστικά αυτών

1.1 ΕΙΣΑΓΩΓΗ

Η συνεχής αυξανόμενη χρήση φορητών συσκευών μικρών διαστάσεων οδήγησε στην ανάπτυξη διαφόρων Λειτουργικών Συστημάτων που φέρουν λειτουργίες ίδιες με αυτές ενός υπολογιστή μεγαλύτερων διαστάσεων. Οι χρήστες τείνουν να χρησιμοποιούν στην καθημερινότητα τους περισσότερο συσκευές όπως smartphone και tablet και όχι παραδοσιακούς υπολογιστές όπως σταθεροί υπολογιστές και laptop. Ωστόσο, λόγω των χαρακτηριστικών των φορητών συσκευών μικρών διαστάσεων (μέγεθος οθόνης, παροχή ενέργειας, ισχύς επεξεργαστή, μέγεθος μνήμης τόσο προσωρινής όσο και μόνιμης), η ανάπτυξη εφαρμογών για αυτές εγκυμονούν πολλές δυσκολίες.

Ακόμα, η ραγδαία αύξηση των δεδομένων που πρέπει να αποθηκευτούν έχουν οδηγήσει στην δημιουργία νέων τεχνολογιών στις Βάσεις Δεδομένων. Πλέον γίνεται ολοένα και μεγαλύτερη η ανάγκη για χρήση Βάσεων Δεδομένων που θα χρησιμοποιούν υπολογιστές χαμηλών τεχνικών χαρακτηριστικών και ταυτόχρονα θα μπορούν να αναπτύσσονται εύκολα χωρίς μεγάλο κόστος.

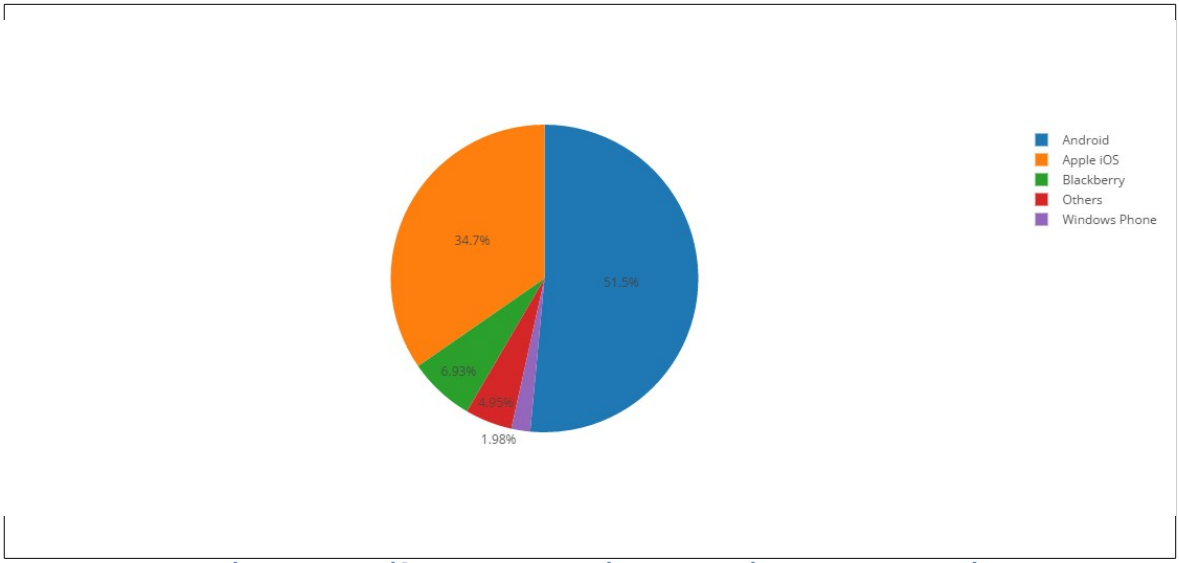
Λαμβάνοντας υπόψιν τα παραπάνω προβήκαμε στην ανάπτυξη μιας εφαρμογής που θα απευθύνεται στις παραπάνω τεχνολογίες. Πιο συγκεκριμένα στο Λειτουργικό Σύστημα Android και την Βάση Δεδομένων DB4O, που αποτελεί μια Αντικειμενοστρεφή επιλογή για NoSQL. Σε αυτό το κεφάλαιο θα αναφέρουμε τις τεχνολογίες που χρησιμοποιήθηκαν, όπως Λειτουργικό Σύστημα, γλώσσα προγραμματισμού, τύπος Βάσης Δεδομένων.

1.1 Android

1.1.1 Λειτουργικό σύστημα Android

Η χρήση των έξυπνων συσκευών, smartphone και tablet, γίνεται όλο και πιο δημοφιλής και αυξάνει δυναμικά την απήχηση του σε ανθρώπους που τα χρησιμοποιούν σε μια πληθώρα εφαρμογών. Ένα από τα σημαντικότερα και ευρέως διαδεδομένα Λειτουργικά Συστήματα για αυτές είναι το Λειτουργικό Σύστημα Android, Εικόνα 1. Το Λειτουργικό Σύστημα Android δημιουργήθηκε από το Open Handset Alliance [6], μερικές από τις εταιρίες του οργανισμού αυτού είναι οι Telecom Italia, Vodafone, T-Mobile, Asus, Acer, LG, Motorola, eBay και υποστηρίζεται από την Google Inc. Αξίζει να αναφερθεί ότι το Android αποτελεί ένα

Open Source Λειτουργικό Σύστημα που δημιουργήθηκε όχι για πώληση και αποκόμιση κέρδους. Οι χρήστες και οι δημιουργοί εφαρμογών έχουν τη δυνατότητα να χρησιμοποιούν τον πηγαίο κώδικα χωρίς κάποια συνδρομή, αλλά μόνο με την εναρμόνιση τους με τους κανόνες που διέπουν την άδεια χρήσης του λειτουργικού συστήματος Android.



Εικόνα 1 "Μερίδιο Λειτουργικών Συστημάτων στην Αγορά"

Σε αντίθεση με άλλα Λειτουργικά Συστήματα για φορητές συσκευές το Android σχεδιάστηκε με κύριο γνώμονα να κάνει όσο το δυνατό πιο εύκολη την επικοινωνία υλικού – λογισμικού, αλλά και λογισμικού – περιβάλλοντος αλληλεπίδρασης με τον χρήστη. Το Λειτουργικό Σύστημα Android βασίζεται στο Linux και χρησιμοποιεί ως βασική γλώσσα προγραμματισμού για τη δημιουργία εφαρμογών την Java με τη χρήση του Android SDK. Το SDK αποτελείτε από ένα ολοκληρωμένο σύνολο εργαλείων ανάπτυξης, που περιλαμβάνει ένα πρόγραμμα εντοπισμού σφαλμάτων, βιβλιοθήκες λογισμικού, ένα προσομοιωτή συσκευής για κινητά που βασίζεται στο QEMU (Quick Emulator), τεκμηρίωση, δείγματα κώδικα, και tutorials. Ωστόσο, για την εκτέλεση προγραμμάτων Java στο Λειτουργικό Σύστημα Android δεν χρησιμοποιείται η Java Virtual Machine (JVM), αλλά τα εικονικά περιβάλλοντα Dalvik Virtual Machine (DVM) για παλαιότερες εκδόσεις Android (έως έκδοση 5.0) και Android Runtime (ART) για τις εκδόσεις 5.0.1 και νεότερες.

Με μια διεπαφή χρήστη που βασίζεται στην άμεση χειραγώγηση, το Android έχει σχεδιαστεί κυρίως για την οθόνη αφής φορητών συσκευών όπως smartphones και tablet. Το Λειτουργικό Σύστημα χρησιμοποιεί εισόδους αφής που αόριστα αντιστοιχούν σε πραγματικές ενέργειες, όπως το σύρσιμο και το τσίμπημα(pinch in/out) για να χειραγωγήσουν αντικείμενα επί της οθόνης, καθώς και ένα εικονικό πληκτρολόγιο. Η απάντηση στην είσοδο του χρήστη έχει σχεδιαστεί για να είναι άμεση και να παρέχει ένα ρευστό περιβάλλον αφής, συχνά με τη χρήση των δυνατοτήτων δόνησης της συσκευής για την παροχή ανάδρασης στο χρήστη. Εσωτερικά υλικά, όπως επιταχυνσιόμετρα, γυροσκόπια και αισθητήρες εγγύτητας

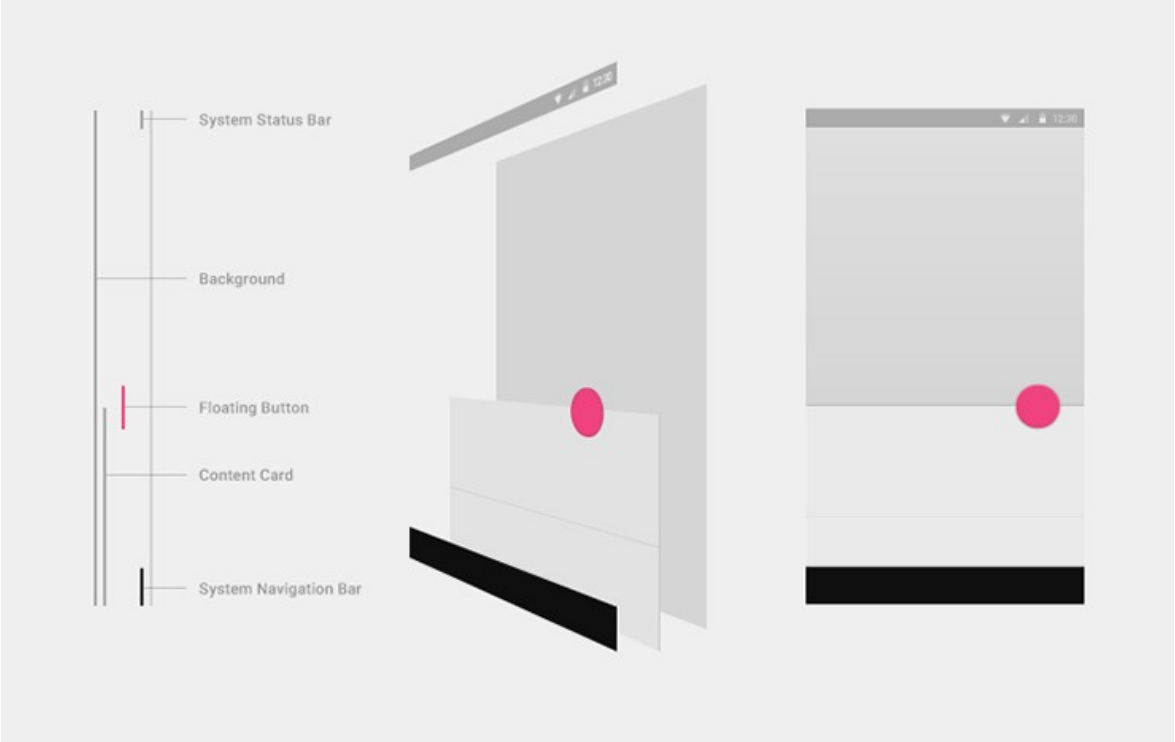
χρησιμοποιούνται από ορισμένες εφαρμογές για να ανταποκριθούν στις πρόσθετες ενέργειες του χρήστη, όπως για παράδειγμα τη ρύθμιση της οθόνης από κατακόρυφη σε οριζόντια ανάλογα με το πώς είναι η συσκευή προσανατολισμένη.[7]

Επειδή οι Android συσκευές συνήθως λειτουργούν με μπαταρία, το Android έχει σχεδιαστεί με στόχο την καλύτερη διαχείριση της μνήμης (RAM) και τη διατήρηση της κατανάλωσης ενέργειας στο ελάχιστο, σε αντίθεση με τα Λειτουργικά Συστήματα που αφορούν Desktop που γενικά υποθέτουν ότι συνδέονται με απεριόριστη πηγή ενέργειας. Όταν μια εφαρμογή Android δεν είναι πλέον σε χρήση, το σύστημα την αναστέλει αυτόματα στη μνήμη ενώ η εφαρμογή εξακολουθεί να είναι τεχνικά "ανοικτή", η εφαρμογή που έχει ανασταλεί δεν καταναλώνει καθόλου πόρους (για παράδειγμα μπαταρία ή ισχύ επεξεργασίας) και θα μείνει σε κατάσταση idle στο παρασκήνιο μέχρι να χρειαστεί και πάλι. Αυτό φέρνει ένα διπλό όφελος από την αύξηση της γενικής ανταπόκρισης των Android συσκευών, δεδομένου ότι η εφαρμογή δεν χρειάζεται να κλείσει και να ξανανοίξει από την αρχή κάθε φορά.[7]

1.1.2 Material Design

Το Material Design, γνωστό και ώς Quantum Paper, αποτελεί ένα σύνολο από κανόνες σχεδίασης (Design language) που παρουσιάστηκε από την Google Inc. στην έκδοση 5.0.0, ωστόσο, με τη χρήση της βιβλιοθήκης v7_appcompact είναι δυνατή η χρήση του Material Design και σε συσκευές που τρέχουν το Λειτουργικό Σύστημα Android 2.1 και νεότερο, με μοναδικό περιορισμό η συσκευή να έχει κατασκευαστεί μετά το 2009. Από τις πρώτες εφαρμογές στις οποίες χρησιμοποιήθηκε ήταν τα γνωστά GAPPS, οι default εφαρμογές της Google που έρχονται προ εγκατεστημένες με κάθε έκδοση του Λειτουργικού Συστήματος Android. ‘Αξιο αναφοράς είναι επίσης το γεγονός ότι το Material Design δεν περιορίζεται μόνο στο Android, αλλά αποτελεί τη βάση για την ανάπτυξη εφαρμογών για όλο το οικοσύστημα της Google Inc. Τέλος, η Google έχει διαθέσει ένα Application Programming Interface (API) έτσι ώστε όποιος προγραμματιστής επιθυμεί να μπορεί να χρησιμοποιεί αυτούς τους κοινούς κανόνες σχεδίασης στην δικιά του εφαρμογή.

Τεχνικά το Material Design είναι ένα τρισδιάστατο περιβάλλον όπου ο χρήστης αλληλεπιδρά με τους άξονες **x**,**y**,**z**, όπου ο άξονας **z** αποτελεί το κομμάτι αυτό που προσδίδει στην εφαρμογή το εφέ της 3^{ης} διάστασης. Η τιμή της μεταβλητής **z** είναι προκαθορισμένη και ισούται με 1 pixel. Μοναδικός περιορισμός του Material Design είναι ότι δεν μπορεί να υπάρξει επικάλυψη από άλλο στρώμα Material Design. [8]



Εικόνα 2 "Χαρακτηριστικά του Material Design"

1.2 Βάση Δεδομένων

1.2.1 NoSQL

Η διαρκής αύξηση της ανάγκης για αποθήκευση όλο και περισσότερων πληροφοριών οδήγησε στην ανάγκη δημιουργίας ενός νέου πεδίου στο χώρο της αποθήκευσης δεδομένων, αυτό είχε σαν αποτέλεσμα την δημιουργία και ανάπτυξη της NoSQL (Not only SQL) το 2009 και γνωρίζει σημαντική άνθιση τα τελευταία χρόνια. Υπάρχουν πάρα πολλά είδη/τύποι NoSQL, με συνολικά τις NoSQL επιλογές να υπερβαίνουν αυτή τη στιγμή τις 225 [9]. Δύο από τις σημαντικότερες κατηγορίες στις οποίες μπορούμε να ομαδοποιήσουμε τις NoSQL Βάσεις είναι οι Column Oriented και οι Object Oriented.

Ένα περιβάλλον Βάσεων Δεδομένων NoSQL αποτελείται από μη-σχεσιακά και σε μεγάλο βαθμό κατανεμημένα Σύστηματα Βάσεων Δεδομένων που επιτρέπουν τη γρήγορη, ad-hoc οργάνωση και την ανάλυση πολύ μεγάλου όγκου και ανόμοιων τύπων δεδομένων. Οι NoSQL Βάσεις Δεδομένων μερικές φορές αναφέρονται ως cloud databases, non-relational databases, Big Data databases. Σε γενικές γραμμές, οι NoSQL Βάσεις Δεδομένων έχουν γίνει η πρώτη εναλλακτική λύση για Σχεσιακές Βάσεις Δεδομένων, με δυνατότητα κλιμάκωσης, διαθεσιμότητα, και ανοχή σε σφάλματα να είναι οι βασικοί λόγοι για την επιλογή της NoSQL. Πηγαίνουν πολύ πιο πέρα από τις πιο ευρέως παραδεκτές παλαιού τύπου,

Σχεσιακές Βάσεις Δεδομένων (όπως η Oracle, ο SQL Server και τη βάση δεδομένων DB2) στην ικανοποίηση των αναγκών των σύγχρονων επιχειρηματικών εφαρμογών. Ένα πολύ ευέλικτο και χωρίς σχήμα μοντέλο δεδομένων, με οριζόντια επεκτασιμότητα, κατανεμημένες αρχιτεκτονικές, και την χρήση γλωσσών και διεπαφών που είναι "Not Only" SQL τυπικά χαρακτηρίζουν αυτήν την τεχνολογία. [10]

Το Big Data είναι μία από τις βασικές κινητήριες δυνάμεις της ανάπτυξης και της αύξησης της δημοτικότητας των Βάσεων Δεδομένων NoSQL. Η σχεδόν ανεξάντλητη ποικιλία τεχνολογιών συλλογής δεδομένων που κυμαίνεται από μια απλή ηλεκτρονική εφαρμογή για συστήματα σημείων πώλησης με τη χρήση GPS σε smartphones και tablets μέχρι εξελιγμένους αισθητήρες και πολλά άλλα. Ένα έργο Big Data συνήθως χαρακτηρίζεται από:

I. ***Υψηλή ταχύτητα δεδομένων*** - πολλά δεδομένα έρχονται πολύ γρήγορα, ενδεχομένως από διάφορες τοποθεσίες.
II. ***Ποικιλία Δεδομένων*** - αποθήκευση των δεδομένων που είναι δομημένα, ημι-δομημένα και μη δομημένα.
III. ***Όγκος δεδομένων*** - τα δεδομένα που περιλαμβάνει είναι πολλά terabytes ή petabytes σε μέγεθος.
IV. ***Πολυπλοκότητα των δεδομένων*** - τα δεδομένα που αποθηκεύονται και διαχειρίζονται σε διαφορετικές τοποθεσίες ή κέντρα δεδομένων. [10]

Το μοναδικό αρνητικό των NoSQL Βάσεων Δεδομένων σε σχέση με τις παραδοσιακές Σχεσιακές Βάσεις Δεδομένων είναι η θυσία της λειτουργικότητας ACID (Atomicity, Consistency, Isolation, Durability) που αποτελεί το βασικό χαρακτηριστικό των RDBMS προς χάρη της διαθεσιμότητας και της επεκτασιμότητας με βασικό χαρακτηριστικό το BASE (Basically Availability soft-state Services with Eventual-consistency) συστήματα.

1.2.2 Column Oriented Database

Οι Βάσεις Δεδομένων Column Oriented έχουν κεντρίσει τα τελευταία χρόνια την προσοχή τόσο της επιστημονικής όσο και της βιομηχανικής κοινότητας. Στις Βάσεις αυτές, τα δεδομένα αποθηκεύονται σε στήλες. Αυτές οι στήλες λειτουργούν ως ευρετήριο της Βάσης. Σημαντικά γνωρίσματα αυτών των βάσεων αποτελούν η ύπαρξη μιας ομοιογένειας στους τύπους δεδομένων με παρόμοια χαρακτηριστικά καθώς και το καλό σύστημα συμπίεσης των δεδομένων. Λόγω του γνωρίσματος των Column Oriented Βάσεων Δεδομένων, της συνάθροισης μεγάλου όγκου δεδομένων, γνωρίζουν μεγάλη απήχηση στον ολοένα και αυξανόμενο χώρο του Διαδικτύου των Πραγμάτων (Internet of Things). Βασικοί εκπρόσωποι αυτής της

κατηγορίας είναι οι Cassandra και HBase, η τελευταία είναι εμπνευσμένη από το Big Table της Google Inc.

1.2.3 Object oriented Database

Οι Object Oriented Βάσεις Δεδομένων ή Object Oriented Database Management System (OODBMS) αποτελούν Βάσεις Δεδομένων στις οποίες τα δεδομένα/πληροφορία αποθηκεύονται/απεικονίζονται ως αντικείμενα. Οι Object Oriented Βάσεις Δεδομένων είναι άκρως ελκυστικές στους αντικειμενοστρεφείς προγραμματιστές, καθώς μπορούν να δημιουργούν αντικείμενα, να τα αποθηκεύουν, να τα τροποποιούν χρησιμοποιώντας την αντικειμενοστρεφή γλώσσα προγραμματισμού της επιλογής τους, δίχως την εξατομικευμένη γνώση των λειτουργειών μιας Βάσης Δεδομένων, όπως θα συνέβαινε σε μια παραδοσιακή Σχεσιακή Βάση Δεδομένων. Ωστόσο, παρά το γεγονός ότι η NoSQL τεχνολογία γνωρίζει άνθιση από το 2009, οι OODBMS βάσεις Δεδομένων υπάρχουν από τα μέσα της δεκαετίας του 1970. Κύριοι εκπρόσωποι αυτής της κατηγορίας αποτελούν τα IRIS της HP, ODE της Bell Labs και Zeitgeist της Texas Instrument.

1.2.4 DB4O

Η db4o [11] είναι μια Object Oriented NoSQL [3] Βάση Δεδομένων. Αποτελεί μια Open Source επιλογή τόσο για Java όσο και για .NET προγραμματιστές. Δημιουργήθηκε το 2000 από τον Carl Rosenberger, το 2008 εξαγοράστηκε από την Versant, εταιρεία του ομίλου Actian. Βασικά χαρακτηριστικά της db4o, εκτός από την native υποστήριξη για Java και .NET, είναι το γεγονός ότι δεν απαιτεί την εγκατάσταση κάποιου πολύπλοκου συστήματος, όπως οι παραδοσιακές Σχεσιακές Βάσεις Δεδομένων ή οι Column Oriented NoSQL, αλλά την ύπαρξη ενός απλού αρχείου, τις περισσότερες φορές με την κατάληξη “.db4o”. Αυτό έχει σαν αποτέλεσμα ένα footprint της τάξης των 670kB για τις .NET εφαρμογές και μόλις 1MB για τις αντίστοιχες Java εφαρμογές. Επιπλέον, αξίζει να σημειωθεί ότι η db4o διαθέτει out-of-the-box υποστήριξη τόσο για server-side όσο και για client-side λειτουργικότητα. Για την ανάπτυξη και δημιουργία μιας db4o Βάσης Δεδομένων δίνεται η δυνατότητα στον προγραμματιστή να χρησιμοποιήσει το Object Management Enterprise (OME) εργαλείο που είναι plug-in για τα IDE Eclipse και Microsoft Visual Studio και επιτρέπει τη δημιουργία και εκτέλεση ερωτημάτων με τη χρήση Γραφικού Περιβάλλοντος. Τέλος, προσφέρεται κάτω από πολλαπλές άδειες χρήσης, συμπεριλαμβανομένης της GNU General Public License (GPL), το db4o Open Source Compatibility License (dOCL), και μια εμπορική άδεια για χρήση σε ιδιόκτητο λογισμικό. [12]

Για τη δημιουργία ενός query υπάρχουν οι παρακάτω τρόποι :

I. ***Query-By-Example:*** Όταν χρησιμοποιείτε Query By Example (QBE) δίνετε στην db4o ένα αντικείμενο πρότυπο. Η db4o θα επιστρέψει όλα τα αντικείμενα που ταιριάζουν με όλες τις μη-προεπιλεγμένες τιμές πεδίου και ενωμένες με τη χρήση του λογικού συνδέσμου “AND”. [2]

II. ***Native Queries***: Τα Native Queries είναι η κύρια διασύνδεση ερωτημάτων της db4o και είναι ο ενδεδειγμένος τρόπος για την αναζήτηση στη βάση δεδομένων από μια εφαρμογή. Επειδή τα Native Queries χρησιμοποιούν μόνο τη σημασιολογία της γλώσσα προγραμματισμού, είναι απόλυτα τυποποιημένα και αποτελούν μια ασφαλή επιλογή για το μέλλον. [2]

III. ***Soda Queries:*** Τα SODA Queries είναι το χαμηλό επίπεδο ερωτημάτων που προσφέρεται από το API της db4o. Κανονικά δεν χρησιμοποιούνται τα SODA, εκτός αν υπάρχουν κάποια ειδικά σενάρια, για παράδειγμα, η δημιουργία ερωτημάτων με δυναμικό τρόπο. Όλα τα άλλα είδη ερωτημάτων, Native Queries και Query-By-Example στην πραγματικότητα μεταφράζονται σε SODA Queries. Για την επίτευξη αυτού η db4o χρησιμοποιεί ανάλυση byte-code και στη συνέχεια δημιουργεί το SODA ερώτημα. Τα απλά αιτήματα μεταφράζονται σε SODA και εκτελούνται απευθείας στη βάση δεδομένων. [13]

Στα κομμάτια κώδικα των Εικόνων 3 έως 5 εκτελέσαμε σχεδόν το ίδιο query με τους τρείς προαναφερθέντες τρόπους. Επειδή η μέθοδος Query-By-Example υποστηρίζει μόνο τον τελεστή **“=”** διαφοροποιείται από τις μεθόδους Native και SODA Queries όπου χρησιμοποιούμε τον τελεστή μεγαλύτερο **“>”** για το πεδίο **κωδικός Αθλητή**. Το αντικείμενο **Αθλητής** αποτελεί αντικείμενο που είναι αποθηκευμένο στη Βάση Δεδομένων που χρησιμοποιήσαμε για την ανάπτυξη της εφαρμογής μας.

```
Athlete ath= new Athlete(8,null,null,null,null,80.0,0,null);
ObjectSet result=db.queryByExample(ath);
```

Εικόνα 3 “Query-By-Example”

```
List<Athlete> result=db.query(new Predicate<Athlete>(){
                      public boolean match(Athlete athlete){
                           return athlete.getCode() > 8
                              &&  athlete.getWeight() = 80.0;
                        }
                       });
```

Εικόνα 4 "Native Query"

```
Query q = db.query();
q.constraint(Athlete.class);
Constraint con = q.descend("code").constraint(8).greater();
q.descend("weight").constraint(80.0).and(con);
ObjectSet result = q.execute();
```

Εικόνα 5 "SODA Query"

1.3 Γλώσσα Προγραμματισμού

1.3.1 Java

Η γλώσσα προγραμματισμού Java δημιουργήθηκε από την Sun Microsystems και παρουσιάστηκε από τον Μάϊο του 1995. Σημαντικά χαρακτηριστικά της Java είναι η απλότητα, η αντικειμενοστρέφεια, το γεγονός ότι το ίδιο εκτελέσιμο μπορεί να εκτελεστεί σε πολλαπλά λειτουργικά συστήματα, η υποστήριξη multi-threading και υψηλής απόδοσης. Ακόμα, η Java αποτελεί την πρώτη γλώσσα προγραμματισμού με built-in δυνατότητες διαδικτυακών εφαρμογών. Ο προγραμματισμός σε Java, όπως και στις περισσότερες αντικειμενοστρεφείς γλώσσες προγραμματισμού, επικεντρώνεται στην ενθυλάκωση τύπων δεδομένων και λειτουργιών για δεδομένα. Το παραπάνω αποτελεί τον ορισμό μιας κλάσης. Οι κλάσεις οργανώνονται σε πακέτα τα οποία καλούνται βιβλιοθήκες. Ένα απο τα δυνατά σημεία της Java είναι η παροχή ενός Application Programming Interface (API), το οποίο προβάλει την επαναχρησιμοποίηση και βοηθάει τους προγραμματιστές από την επαναδημιουργία κώδικα που ήδη λειτουργεί επιτυχώς.

Ένα άλλο χαρακτηριστικό που προσφέρει η Java είναι το Γραφικό Περιβάλλον χρήστη. Το πλεονέκτημα αυτής της γλώσσας προγραμματισμού είναι ότι το ίδιο Graphical User Interface (GUI) εκτελείται χωρίς πρόβλημα σε διάφορα περιβάλλοντα, όπως Windows, Linux, Mac OS. Αυτό επιτρέπει στους προγραμματιστές να δημιουργούν γραφικό περιβάλλον με buttons, text areas,

textfields, labels, για την αλληλεπίδραση του χρήστη με αυτά τόσο με τη χρήση ενός ποντικιού και πληκτρολογίου όσο και με την αλληλεπίδραση του χρήστη με το πρόγραμμα μέσω οθόνης αφής, αυξάνοντας την διαδραστικότητα με τον χρήστη.

1.3.2 Τρόπος λειτουργίας του Garbage Collector στη Java

Το Java Memory Management, με τον native Garbage Collector είναι ένα από τα σημαντικότερα χαρακτηριστικά αυτής της γλώσσας προγραμματισμού. Αυτό επιτρέπει στους προγραμματιστές να δημιουργούν αντικείμενα χωρίς να ανησυχούν για τη διαχείριση της μνήμης που καταναλώνεται από το εκάστοτε πρόγραμμα, καθώς ο Garbage Collector ελευθερώνει χώρο από τη μνήμη για την καλύτερη δυνατή επαναχρησιμοποίηση της. Αποτέλεσμα αυτού είναι η επίλυση πάρα πολλών προβλημάτων που αντιμετώπιζαν οι προγραμματιστές για τη διαχείριση της μνήμης.

Ο Garbage Collector εντοπίζει τα αντικείμενα που είναι ενεργά και μαρκάρει όλα τα υπόλοιπα ως «σκουπίδια». Για να αποφασίσει ποιά αντικείμενα δεν χρησιμοποιούνται, η Java Virtual Machine εκτελεί ανα τακτά χρονικά διαστήματα τον αλγόριθμο «mark-and-sweep».

Τα βήματα που εκτελούνται είναι τα παρακάτω :

⇒ Ο αλγόριθμος ελέγχει όλες τις αναφορές αντικειμένων ξεκινώντας από τα GC Roots και σημαίνοντας όλα τα αντικείμενα που βρίσκει ώς ενεργά.
⇒ Τα κομμάτια της heap memory που είναι δεσμευμένα από αντικείμενα που δεν έχουν τη σήμανση ενεργά, απελευθερώνονται διαγράφοντας τα.

Συνοψίζοντας, ο Garbage Collector δημιουργήθηκε για την εξάλειψη του φαινομένου υπερφόρτωσης της μνήμης, πράγμα το οποίο δημιουργείται με αντικείμενα που δεν έχουν διαγραφεί και δεν χρησιμοποιούνται από την εφαρμογή ή δεν υπάρχουν αναφορές τους μέσα στο πρόγραμμα, σε αυτή την περίπτωση ο προγραμματιστής δεν κατέστρεψε τα αντικείμενα αυτά. [14]

ΕΠΙΛΟΓΟΣ

Η πτυχιακή εργασία θα βασιστεί πάνω σε αυτές τις τεχνολογίες. Το Android είναι ένα Λειτουργικό Σύστημα που δημιουργήθηκε για κινητές συσκευές έτσι ώστε να γίνεται η καλύτερη δυνατή διαχείριση των μειωμένων πόρων που διαθέτει καθώς και την πολυπλοκότητα του με την είσοδο αφής. Στον τομέα της αποθήκευσης δεδομένων η ολοένα και αυξανόμενη απαίτηση των εφαρμογών για μεγαλύτερη ελευθερία στο μέγεθος και στη δομή των Βάσεων Δεδομένων οδήγησε στην ανάπτυξη της NoSQL, δηλαδή Βάσεων Δεδομένων που επεκτείνουν την

λειτουργικότητα των υπαρχουσών βάσεων δεδομένων SQL. Σε αυτόν τον τομέα της NoSQL σημαντικό παράγοντα έπαιξαν οι αντικειμενοστρεφείς βάσεις δεδομένων, όπως η DB4O, που δίνουν τη δυνατότητα στον προγραμματιστή να αποθηκεύει αντικείμενα της γλώσσας προγραμματισμού που χρησιμοποιεί. Στο επόμενο κεφάλαιο θα γίνει αναφορά του προγράμματος εξυπηρετητή και πώς χρησιμοποιήθηκαν οι παραπάνω τεχνολογίες.

ΚΕΦΑΛΑΙΟ 2

Πρόγραμμα Εξυπηρετητή για DB4O

ΕΙΣΑΓΩΓΗ

Για τη δοκιμή της εφαρμογής μας χρειάστηκε να κατασκευάσουμε ένα πρόγραμμα το οποίο θα λειτουργεί ως εξυπηρετητής για τη Βάση Δεδομένων db4o. Το πρόγραμμα αυτό επιλέχθηκε να κατασκευαστεί στη γλώσσα προγραμματισμού Java, καθώς αυτό μας επιτρέπει την εγκατάστασή του σε πληθώρα Λειτουργικών Συστημάτων (Windows, Linux, Mac OS). Επιπλέον, το γεγονός ότι η Java διαθέτει build-in Garbage Collector αποτελεί έναν ακόμα λόγο για την επιλογή της καθώς επιτρέπει στον προγραμματιστή να δημιουργεί πολλαπλά νέα αντικείμενα χωρίς να ανησυχεί για την σωστή και αποδοτικότερη διαχείριση της μνήμης. Αυτό αποτελεί σημαντικό κομμάτι στην δημιουργία αυτής της εφαρμογής, καθώς μέσα από επιστημονική έρευνα που διενεργήθηκε για το όσο το δυνατό ακριβέστερο προσδιορισμό του footprint της εφαρμογής εξυπηρετητή και η μεγαλύτερη δυνατή μείωση του.

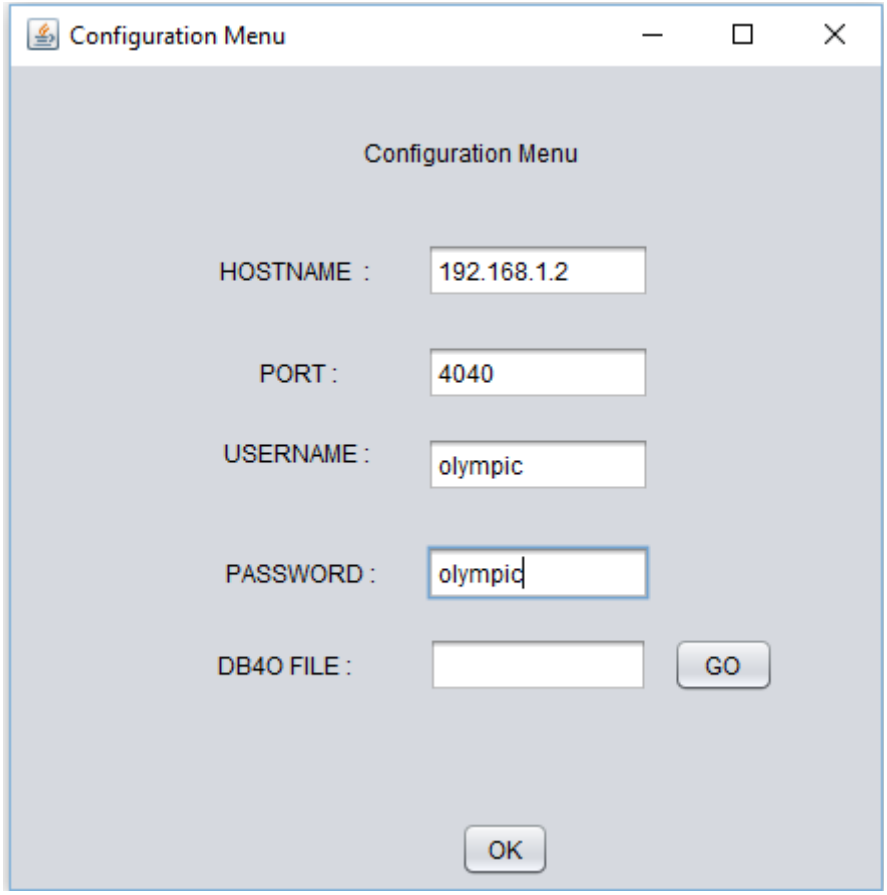
2.1 Γραφικό Περιβάλλον Εξυπηρετητή

Κύριος στόχος της εφαρμογής Εξυπηρετητή είναι η παροχή ενός απλού Γραφικού Περιβάλλοντος χρήστη καθιστώντας το έτσι χρησιμοποιήσιμο από χρήστη δίχως ενδελεχείς γνώσεις στην Επιστήμη της Πληροφορικής. Όπως αναφέρθηκε στο υποκεφάλαιο 1.3.1, η γλώσσα προγραμματισμού Java προσφέρει ένα κοινό Γραφικό Περιβάλλον χρήστη ανεξαρτήτως Λειτουργικού Συστήματος, με τις μόνες αλλαγές να εντοπίζονται στην εμφάνιση των λειτουργιών σμίκρυνση/μεγέθυνση, κλείσιμο, κάθε παραθύρου που κληρονομείται από το εκάστοτε Λειτουργικό Σύστημα. Μέσα από τις βιβλιοθήκες Γραφικού Περιβάλλοντος που διαθέτει η Java μας δίνεται η δυνατότητα να δημιουργούμε παράθυρα με buttons, textfields, text areas και πληθώρα άλλων γραφικών χαρακτηριστικών που αυξάνουν τη διαδρ

αστικότητα με τον χρήστη, καθιστώντας την είσοδο και έξοδο του προγράμματος περισσότερο οικείο και αποδοτικό σε σχέση με το μη Γραφικό Περιβάλλον για τον χρήστη.

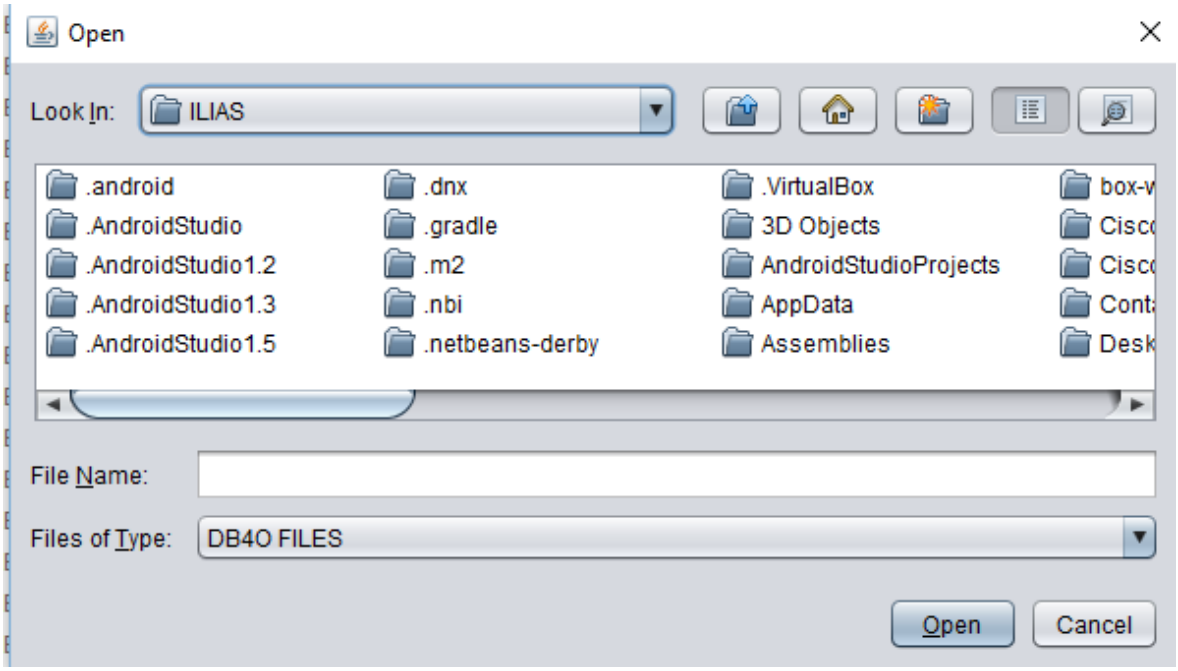
Ο χρήστης εκκινώντας την εφαρμογή Εξυπηρετητή έρχεται σε επαφή με τη αρχική φόρμα, Εικόνα 6. Η φόρμα αυτή αποτελείται από τρία χαρακτηριστικά, δύο κουμπιά και μία εικόνα. Το πρώτο κουμπί, με όνομα **“Configure”**, βρίσκεται στην πάνω αριστερά πλευρά του παραθύρου και μπορεί να χρησιμοποιηθεί από τον χρήστη από την έναρξη της εφαρμογής. Με την επιλογή αυτού του κουμπιού από τον χρήστη με την ιδιότητα Action Performed. Στη συνέχεια και εφόσον ο χρήστης επιλέξει το κουμπί **“Configure”** μεταβαίνει στην επόμενη Φόρμα, Εικόνα 7. Εκ δεξιόν του χαρακτηριστικού κουμπιού με όνομα **“Configure”** βρίσκεται το κουμπί **“startServer”**. Σε αντίθεση με το πρώτο κουμπί της φόρμας αυτής, το κουμπί **“startServer”** δεν είναι επιλέξιμο από την έναρξή της αλλά μόνο μετά την επιλογή του κουμπιού **“Configure”**, τουλάχιστον μία φορά. Με αυτό τον τρόπο κατοχυρώνουμε ότι ο χρήστης δεν θα εκκινήσει τον Εξυπηρετητή δίχως να έχει θέσει τα αναγκαία χαρακτηριστικά όπως τη διεύθυνση, την πόρτα στην οποία θα αναμένει αιτήματα, το όνομα χρήστη και τον κωδικό που χρησιμεύουν μόνο στην απομακρυσμένη σύνδεση. Μετά την επιλογή αυτού του στοιχείου εκκινείται ο Εξυπηρετητής db4o και μπορούμε να συνδεθούμε απομακρυσμένα σε αυτόν. Επόμενο γραφικό στοιχείο της αρχικής φόρμας της εφαρμογής αυτής είναι ένα γραφικό στοιχείο **“Image”** το οποίο βρίσκεται κάτω από τα προαναφερθέντα γραφικά στοιχεία “κουμπιά” και στοιχισμένο στο κέντρο του παραθύρου, στόχος του οποίου είναι η βελτίωση του γραφικού περιβάλλοντος για τον χρήστη καθώς πλέον δεν αποτελείται μόνο από λειτουργίες (κουμπιά, πεδία συμπλήρωσης κειμένου).

Εικόνα 6 "Αρχική Φόρμα"



Εικόνα 7 "Φόρμα πληροφοριών Εξυπηρετητή"

Η φόρμα **“Configuration Menu”** αποτελείται από έξι labels, πέντε textfields και δύο κουμπιά. Το πρώτο label βρίσκεται στο κέντρο του παραθύρου στο πάνω μέρος αυτής με τίτλο **“Configuration Menu”**. Στη συνέχεια τα υπόλοιπα πέντε label βρίσκονται στοιχισμένα εκ αριστερόν των πέντε textfields, προσφέροντας έτσι πληροφορία για το τι θα πρέπει να συμπληρώσει στα αντίστοιχα textfields ο χρήστης. Τα labels βρίσκονται εκ αριστερόν καθώς αυτός είναι ο πλέον κοινά αποδεκτός τρόπος απεικόνισης φορμών, δίχως να επηρεάζεται το “User Experience” του χρήστη. Στο τελευταίο textfield, δεξιά από το label με τίτλο **“DB4O FILE”**, ο χρήστης δεν δύναται να πληκτρολογήσει κάποια τιμή αλλά επιλέγοντας το κουπί **“GO”** που βρίσκεται εκ δεξιόν αυτού ανοίγει ένας περιηγητής αρχείων, Εικόνα 8. Με τη χρήση αυτού του περιηγητή αρχείων ο χρήστης μπορεί να επιλέξει το db4o αρχείο που θέλει να χρησιμοποιήσει ως Βάση Δεδομένων και να την κάνει προσβάσιμη μέσω δικτύου, όπως μπορούμε να δούμε στην Εικόνα 8 ο τύπος των αρχείων που μπορούν να επιλεγούν είναι .db4o μειώνοντας ή εξαλείφοντας έτσι το ενδεχόμενο ο χρήστης να επιλέξει λανθασμένο τύπο αρχείου. Αφού ο χρήστης έχει συμπληρώσει τα πεδία αυτής της φόρμας και επιλέγοντας το γραφικό στοιχείο κουμπί **“OK”** που βρίσκεται στο κάτω μέρος του παραθύρου και είναι στοιχισμένο στο κέντρο, επιστρέφει στο αρχικό παράθυρο, αφού κλείσει το παρόν παράθυρο, μπορεί να εκκινήσει τον εξυπηρετητή.



Εικόνα 8 “Φόρμα περιηγητή αρχείων”

2.2 Κώδικας για την υλοποίηση του Εξυπηρετητή

Σε αυτό το υποκεφάλαιο θα δούμε τα βασικά κομμάτια κώδικα που αφορούν την υλοποίηση Εξυπηρετητή για τη Βάση Δεδομένων db4o. Οι βασικές κλάσεις που υλοποιούν αυτή τη λειτουργικότητα είναι οι: DB4O και StartServer.

Η κλάση DB4O, αποτελεί την φόρμα, Γραφικό Περιβάλλον της εφαρμογής με την οποία ο χρήστης αλληλεπιδρά. Είναι υπεύθυνη για την διαχείριση και τη λειτουργικότητα των διάφορων components που χρησιμοποιούμε ως είσοδο και έξοδο του προγράμματός μας, όπως είδαμε στο υποκεφάλαιο 2.1. Στον κώδικα της Εικόνας 9 στη γραμμή ένα βλέπουμε την μέθοδο **"StartServerActionPerformed"**, αυτή η μέθοδος καλείται κάθε φορά που ο χρήστης προβαίνει σε κάποια ενέργεια με το συγκεκριμένο component ανεξαρτήτως εισόδου, μπορεί να είναι ένα ποντίκι, ένα πλήκτρο πληκτρολογίου ή και μία οθόνη αφής. Στη γραμμή δύο δημιουργούμε ένα αντικείμενο τύπου **ExecutorService**, το οποίο λαμβάνει ένα κομμάτι από την λίστα των διαθέσιμων διεργασιών. Αυτό συμβαίνει καθώς ο Εξυπηρετητής αποτελεί μια multi-threading εφαρμογή για την παράλληλη χρήση πολλαπλών αρχείων db4o, όχι όμως του ίδιου αρχείου db4o καθώς το εκάστοτε db4o αρχείο "κλειδώνεται" και δεν μπορεί να χρησιμοποιηθεί από οποιαδήποτε άλλη διεργασία. Στη γραμμή τρία δημιουργούμε ένα νέο αντικείμενο τύπου **StartServer** περνώντας τις απαραίτητες παραμέτρους που έχουμε συλλέξει από τον χρήστη μέσα από την αντίστοιχη Φόρμα, Εικόνα 7.

```
1.  private void StartServerActionPerformed(java.awt.event.ActionEvent evt) {
2.      ExecutorService executor = Executors.newFixedThreadPool(30);
3.      Runnable ss=new StartServer(host,file,port,uname,passwd);
4.      executor.execute(ss);
5.  }
```

Εικόνα 9 "Μέθοδος StartServerActionPerformed"

Όπως φαίνεται στον κώδικα της Εικόνας 10, η **StartServer** υλοποιεί το Interface **Runnable** που χρησιμοποιείται για multi-thread προγραμματισμό. Στη συνέχεια, γραμμή τέσσερα, εκτελούμε ως διεργασία στο παρασκήνιο το αντικείμενο **StartServer** που δημιουργήσαμε στη γραμμή τρία. Στην κλάση **StartServer**, ο κώδικας της Εικόνας 11, έχουμε την μέθοδο **"run"** που αποτελεί την κύρια μέθοδο ενός αντικειμένου τύπου **Runnable**. Στις γραμμές τρία και τέσσερα, δημιουργούμε ένα αντικείμενο τύπου **ServerConfiguration** όπου μπορούμε να αποθηκεύσουμε τις απαραίτητες ρυθμίσεις για την db4o, ανεξαρτήτως αν χρησιμοποιούμε απομακρυσμένη ή τοπική Βάση Δεδομένων db40 πρέπει να ορίσουμε ένα ήδη υπάρχον αρχείο ρυθμίσεων ή να δημιουργήσουμε ένα νέο, όπως κάνουμε σε

αυτές τις γραμμές. Στις γραμμές οχτώ έως δέκα, δημιουργούμε ένα αντικείμενο τύπου **ObjectServer**, που δέχεται ώς παραμέτρους το αρχείο ρυθμίσεων που δημιουργήσαμε νωρίτερα, το αρχείο db4o που θα κάνουμε διαθέσιμο μέσω του Εξυπηρετητή και τέλος την πόρτα στην οποία ο Εξυπηρετητής θα δέχεται ερωτήματα. Στη γραμμή έντεκα θέτουμε τους χρήστες που έχουν πρόσβαση στον πόρο, αρχείο db4o, που μόλις κοινοποιήσαμε. Με τον ίδιο τρόπο θα μπορούσαμε να ορίσουμε πολλαπλούς χρήστες με τα ίδια ή και διαφορετικά δικαιώματα στο db4o αρχείο. Στις γραμμές δεκατέσσερα έως δεκαεφτά ορίζουμε το όνομα του Thread, τόσο για την διαδικασία της αποσφαλμάτωσης όσο και στην περίπτωση που θελήσουμε να παύσουμε την εκτέλεση του και ορίζουμε την προταιρεότητα του σε σχέση με τα υπόλοιπα Thread που εκτελούνται. Στις γραμμές δεκαοχτώ έως είκοσι δύο έχουμε έναν ατέρμονα βρόγχο ο οποίος μπορεί να τερματιστεί μόνο αν στείλουμε κάποιο μήνυμα τερματισμού για το συγκεκριμένο Thread του Εξυπηρετητή ή αν προκύψει κάποιο σφάλμα που το τερματίσει βίαια. Εάν υπάρξει τερματισμός του Thread είτε επιδιωκόμενος είτε βίαιος θα πρέπει να τερματιστεί και η τοπική σύνδεση, κλείδωμα, του αρχείου db4o για τη μελλοντική χρήση του αρχείου, όπως φαίνεται στην γραμμή είκοσι έξι.

```
1. public class StartServer implements Runnable
```

Εικόνα 10 "Κλάση StartServer"

```
1.  public void run() {
2.          synchronized (this) {
3.              ServerConfiguration config =
4.          Db4oClientServer.newServerConfiguration();
5.              // Using the messaging functionality to redirect all
6.              // messages to this.processMessage
7.             // config.networking().messageRecipient(this);
8.              ObjectServer db4oServer =
9.          Db4oClientServer.openServer(config
10.                     , FILE, PORT);
11.             db4oServer.grantAccess(USER, PASS);
12.
13.             // to identify the thread in a debugger
14.         Thread.currentThread().setName(this.getClass().getName());
15.          // We only need low priority since the db4o server has
16.             // it's own thread.
17.             Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
18.             try {
19.                 if (!stop) {
20.                     // wait forever for notify() from close()
21.                     this.wait(Long.MAX_VALUE);
22.                 }
23.             } catch (Exception e) {
24.                 e.printStackTrace();
25.             }
26.             db4oServer.close();
27.         }
28.     }
```

Εικόνα 11 "Μέθοδος run()"

ΕΠΙΛΟΓΟΣ

Σε αυτό το κεφάλαιο είδαμε ένα πρόγραμμα Εξυπηρετητή για τη Βάση Δεδομένων db4o που δημιουργήσαμε για την αποσφαλμάτωση και την ανάπτυξη του προγράμματος απομακρυσμένης σύνδεσης με Βάση Δεδομένων db4o για συσκευές με το Λειτουργικό Σύστημα Android. Έγινε αναφορά τόσο στο Γραφικό Περιβάλλον χρήστη όσο και στα βασικά σημεία του κώδικα, που επιτρέπουν στον χρήστη να ορίσει τις βασικές παραμέτρους που απαιτούνται για την λειτουργία ενός Εξυπηρετητή για τη Βάση Δεδομένων db4o. Στο επόμενο κεφάλαιο θα δούμε τον τρόπο που ένα πρόγραμμα Πελάτης προσπελαύνει την πληροφορία που είναι διαθέσιμη από ένα πρόγραμμα Εξυπηρετητή.

ΚΕΦΑΛΑΙΟ 3

Πρόγραμμα Πελάτη για DB4O (Για Συσκευές Android)

ΕΙΣΑΓΩΓΗ

Η διαρκής αύξηση της χρήσης φορητών συσκευών περιορισμένων πόρων (smartphones, tablets) και η παράλληλη εξάπλωση της χρήσης Βάσεων Δεδομένων τύπου NoSQL μας οδήγησαν στην δημιουργία ενός προγράμματος Πελάτη για το Λειτουργικό Σύστημα Android και τη Βάση Δεδομένων db4o. Θα ακολουθήσει μία λεπτομερής περιήγηση στα βασικά αρχεία που χρειαστήκαμε, πρώτο υποκεφάλαιο, για την υλοποίηση της εφαρμογής Πελάτη και στα δύο επόμενα υποκεφάλαια θα δούμε το Γραφικό Περιβάλλον χρήστη και τον κώδικα πίσω από αυτό αντίστοιχα.

3.1 Βοηθητικές κλάσεις

3.1.1 Db4oSubClass.java

Η κλάση Db4oSubClass αποτελεί μία βοηθητική κλάση που στόχο έχει να συγκεντρώσει και να υλοποιήσει τη λειτουργικότητα που επαναλαμβάνεται σε όλες τις Activities, αφήνοντας σε αυτές μόνο την επεξεργασία των αποτελεσμάτων κατά το δοκούν. Βασική ευθύνη της κλάσης αυτής είναι το άνοιγμα της σύνδεσης και το κλείσιμο αυτής, καθώς και την κοινοποίηση της μεταβλητής **db** του τύπου **ObjectContainer** και στις υπόλοιπες κλάσεις της εφαρμογής με τη χρήση της μεθόδου **“public ObjectContainer getDB()”**, κώδικας της Εικόνας 12.

```
1. public class Db4oSubClass {
2.     private ObjectContainer db;
3.
4.     public ObjectContainer getDb() {
5.         return db;
6.     }
7.
```

Εικόνα 12 “Κλάση Db4oSubClass”

Στη συνέχεια έχουμε μια σειρά από μεθόδους που αφορούν την επιστροφή αποτελεσμάτων με τη χρήση μεταδεδομένων (metadata) τόσο για τις κλάσεις που είναι αποθηκευμένες στη Βάση Δεδομένων db4o που έχει συνδεθεί ο χρήστης,

όσο και για τα πεδία (fields) των κλάσεων αυτών. Η μέθοδος **“public ReflectClass[] reflectClasses()”**, κώδικας της Εικόνας 13, επιστρέφει όλες τις κλάσεις που είναι αποθηκευμένες στην db4o Βάση Δεδομένων ως πίνακα αντικειμένων τύπου **ReflectClass**. Σε αυτές περιλαμβάνονται εκτός από τις κλάσεις-αντικείμενα που έχει αποθηκεύσει και θέλει να προσπελάσει ο χρήστης και οι βοηθητικές κλάσεις που αφορούν τους τύπους των μεταβλητών που έχουν αποθηκευτεί καθώς και κλάσεις που αφορούν την εκτέλεση των ερωτημάτων(Query), μερικές από αυτές είναι οι java.util.ArrayList, java.util.Date, com.db4o.internal.query.processor.QQuery, com.db4o.internal.query.processor.QEContains, οι κλάσεις αυτές είναι ενδεικτικές και μεταβάλλονται ανάλογα με το αν έχει χρησιμοποιηθεί ο τύπος ArrayList και ο τύπος ερωτήματος SODA αντίστοιχα, Εικόνα 14.

```
1. public ReflectClass[] reflectClasses() {
2.     return db.ext().reflector().knownClasses();
3. }
```

Εικόνα 13 “Μέθοδος reflectClasses”

- com.db4o.foundation.Collection4
- com.db4o.foundation.List4
- com.db4o.internal.query.processor.QCon
- com.db4o.internal.query.processor.QConClass
- com.db4o.internal.query.processor.QConObject
- com.db4o.internal.query.processor.QConPath
- com.db4o.internal.query.processor.QE
- com.db4o.internal.query.processor.QEAbstract
- com.db4o.internal.query.processor.QEContains
- com.db4o.internal.query.processor.QEEqual
- com.db4o.internal.query.processor.QEGreater
- com.db4o.internal.query.processor.QEStringCmp
- com.db4o.internal.query.processor.QField
- com.db4o.internal.query.processor.QQuery
- com.db4o.internal.query.processor.QQueryBase
- java.sql.Date
- java.sql.Time
- java.util.AbstractCollection
- java.util.AbstractList
- java.util.ArrayList
- java.util.Date

Εικόνα 14 “Βοηθητικές κλάσεις db4o”

Επόμενη μέθοδος της κλάσης Db4oSubClass είναι η **“public ReflectClass reflectClass(String className)”**, κώδικας της Εικόνας 15,που δέχεται ως είσοδο το όνομα το όνομα της κλάσης που μας ενδιαφέρει ως αλφαριθμητικό και μας την επιστρέφει ως αντικείμενο τύπου **ReflectClass**.

```
1. public ReflectClass reflectClass(String className) {
2.     return db.ext().reflector().forName(className);
3. }
```

Εικόνα 15 “Μέθοδος reflectClass(String className)”

Η μέθοδος **“public List<ReflectClass> reflectClassesASREFC()”**, κώδικας της Εικόνας 16, επιστρέφει μία λίστα αντικειμένων τύπου **ReflectClass**. Η διαφορά της με την **“public ReflectClass[] reflectClasses()”** εκτός από τη διαφορά στον τύπο **“Collection”** είναι το γεγονός ότι στην μέθοδο **“public List<ReflectClass> reflectClassesASREFC()”** και πιο συγκεκριμένα στη γραμμή πέντε πραγματοποιήται ο εξής έλεγχος, να αποθηκεύονται στη λίστα μόνο οι κλάσεις που δεν αποτελούν βοηθητικές κλάσεις, όπως αναφέρθηκε παραπάνω. Ακόμα, επειδή κατά τη διάρκεια των δοκιμών παρατηρήθηκε ότι πολλές φορές η μέθοδος **“public ReflectClass[] reflectClasses()”**, που καλείται και από την μέθοδο **“public List<ReflectClass> reflectClassesASREFC()”**, επιστρέφει παραπάνω από μία φορές την ίδια τιμή ελέγχουμε αν υπάρχει ήδη η συγκεκριμένη κλάση μέσα στη λίστα, το τελευταίο καθώς και το γεγονός ότι ο τύπος λίστα αποτελεί ένα δυναμικό τύπο δεδομένων μας οδήγησαν στην επιλογή αυτού του τύπου σε σχέση με κάποιο άλλο τύπο τύπου **“Collection”**.

```
1. public List<ReflectClass> reflectClassesASREFC() {
2.     List<ReflectClass> knownClasses = new ArrayList<>();
3.     ReflectClass[] sdmkd = reflectClasses();
4.     for (int i = 0; i < sdmkd.length; i++) {
5.         if (!sdmkd[i].toString().contains("com.") &&
   !sdmkd[i].toString().contains("java.") &&
   !knownClasses.contains(sdmkd[i])) {
6.             knownClasses.add(sdmkd[i]);
7.         }
8.     }
9.     return knownClasses;
10.    }
```

Εικόνα 16 “Μέθοδος reflectClassesAsREFC”

Η μέθοδος **“public List<String> reflectClassesAsSTR()”**, κώδικας της Εικόνας 17, επιστρέφει μία λίστα τύπου αλφαριθμητικό και περιέχει τα αποτελέσματα της μεθόδου **“public List<ReflectClass> reflectClassesASREFC()”**. Η μέθοδος αυτή είναι σημαντική καθώς λαμβάνει αντικείμενα τύπου **ReflectClass** και αφού πάρει το όνομα του αντικειμένου το αποθηκεύει σε μία λίστα, έτσι ώστε στη συνέχεια να χρησιμοποιηθεί για το γέμισμα του Recycle View.

```
1. public List<String> reflectClassesAsSTR() {
2.     List<String> reflectATTList = new ArrayList<>();
3.     for (ReflectClass RC : reflectClassesASREFC()) {
4.         reflectATTList.add(RC.getName());
5.     }
6.     return reflectATTList;
7. }
```

Εικόνα 17 “Μέθοδος reflectClassesAsSTR”

Η μέθοδος **“public ReflectField[] reflectFields(String classname)”**, κώδικας της Εικόνας 18, επιστρέφει σε μορφή πίνακα τύπου **ReflectField** όλα τα πεδία (fields) της κλάσης που περάσαμε παραμετρικά σε μορφή αλφαριθμητικού.

```
1. public ReflectField[] reflectFields(String classname) {
2.     return
   db.ext().reflector().forName(classname).getDelegate().getDecl
   aredFields();
3. }
```

Εικόνα 18 “Μέθοδος reflectFields”

Η μέθοδος **“public List<ReflectField> reflectFieldsNameASRF(String className)”**, κώδικας της Εικόνας 19, επιστρέφει μία λίστα τύπου **ReflectField** και δέχεται ως είσοδο το όνομα της κλάσης που μας ενδιαφέρει ως αλφαριθμητικό. Σε αυτή μέθοδο αποθηκεύουμε σε μία λίστα όλα εκείνα τα πεδία που δεν είναι της μορφής X.Object, γραμμή 5, στη θέση του X μπορεί να είναι της μορφής java.util, com.db4o, κλπ.

```
1. public List<ReflectField> reflectFieldsNameASRF(String
   className) {
2.     List<ReflectField> reflectFields = new ArrayList<>();
3.     ReflectField[] allReflectFields =
   reflectFields(className);
4.     for (ReflectField reflectField : allReflectFields) {
5.         if
   (!reflectField.getFieldType().getName().contains(".Object"))
   {
6.             reflectFields.add(reflectField);
7.         }
8.     }
9.     return reflectFields;
10.    }
```

Εικόνα 19 "Μέθοδος reflectFieldsNameASFR"

Η μέθοδος **"public List<String> reflectFieldsNameASSTR(String className)"**, κώδικας της Εικόνας 20, επιστρέφει μία λίστα τύπου αλφαριθμητικό και δέχεται ως είσοδο το όνομα της κλάσης που μας ενδιαφέρει ως αλφαριθμητικό. Όπως και η μέθοδος **"public List<String> reflectClassesAsSTR()"** έτσι και η **"public List<String> reflectFieldsNameASSTR(String className)"** χρησιμοποιείται για το γέμισμα του Recycle View.

```
1. public List<String> reflectFieldsNameASSTR(String className)
   {
2.     List<String> reflectFields = new ArrayList<>();
3.
4.     for (ReflectField reflectField :
   reflectFieldsNameASRF(className)) {
5.         reflectFields.add(reflectField.getName());
6.     }
7.     return reflectFields;
8. }
```

Εικόνα 20 "Μέθοδος reflectFieldsNameASSTR"

Η μέθοδος **"public List<String> reflectFieldsNameANDTypeListSTRING(String className)"**, κώδικας της Εικόνας 21, επιστρέφει μία λίστα τύπου αλφαριθμητικό και δέχεται ως είσοδο το όνομα της κλάσης που μας ενδιαφέρει ως αλφαριθμητικό. Ευθύνη της μεθόδου αυτής είναι να ελέγξει όλα τα πεδία της κλάσης και στη συνέχεια να εμφανίσει τόσο

το όνομα του πεδίου όσο και τον τύπο αυτού. Με τον έλεγχο που πραγματοποιείται στη γραμμή πέντε διαχωρίζουμε τα αντικείμενα **“Collection”** από τα υπόλοιπα αντικείμενα. Αντικείμενα Collection αποτελούν τα AbstractCollection, AbstractList, AbstractSequentialList, LinkedList, ArrayList, AbstractSet, HashSet, LinkedHashSet, TreeSet, AbstractMap, HashMap, TreeMap, WeakHashMap, LinkedHashMap, IdentityHashMap, Vector, Stack, Dictionary, Hashtable, Properties, BitSet [15].

```
1. public List<String> reflectFieldsNameANDTypeListSTRING(String
   className) {
2.    List<String> reflectATTList = new ArrayList<>();
3.    ReflectField[] fields = reflectFields(className);
4.    for (ReflectField rff : fields) {
5.        if (rff.getFieldType().isCollection()) {
6.            reflectATTList.add(rff.getName() + "-->" + "
   isCollection");
7.        } else {
8.            reflectATTList.add(rff.getName() + "-->" +
   rff.getFieldType().getName());
9.        }
10.     }
11.     return reflectATTList;
12.  }
13.
```

Εικόνα 21 “Μέθοδος reflectFieldsNameANDTypeListSTRING”

Η μέθοδος **“public Db4oSubClass(Context ctx)”**, κώδικας της Εικόνας 22, αποτελεί τον δομητή της **Db4oSubClass**. Η μέθοδος αυτή δεν επιστρέφει κάποια τιμή όπως διακρίνουμε από την υπογραφή της και δέχεται ως είσοδο παραμετρικά μία μεταβλητή τύπου **Context**. Το Context είναι μια διεπαφή που επιτρέπει την χρήση διαφόρων συστατικών της εφαρμογής σε διάφορες κλάσεις [16]. Μερικές από τις λειτουργίες της χρήσης αντικειμένου Context είναι η προσπέλαση αρχείων και κλάσεων στα οποία έχει πρόσβαση μόνο η συγκεκριμένη εφαρμογή, η εκκίνηση διεργασίας (Activity). Στη συγκεκριμένη περίπτωση χρησιμοποιείται για να έχουμε πρόσβαση τόσο στα στοιχεία **SharedPreferences** όσο και στη μέθοδο **“getString()”** για τη χρήση των αλφαριθμητικών που βρίσκονται στο αρχείο **“res/values/strings.xml”**. Τα SharedPreferences αποτελούν δείκτες σε αρχεία που περιέχουν ζεύγη κλειδιών-τιμών και μας προσφέρει μεθόδους για την εγγραφή και ανάκτηση αυτών [17]. Τα SharedPreferences ενδείκνυνται σε περιπτώσεις που χρειάζεται να αποθηκεύσουμε μικρό αριθμό από ζεύγη κλειδιών-τιμών, όπως στην περίπτωσή μας που αποθηκεύουμε τα όνομα χρήστη, κωδικό, διεύθυνση εξυπηρετητή και θύρα στην οποία ο εξυπηρετητής αναμένει ερωτήματα. Αρχικά

διαβάζουμε το αρχείο **“DB4OCREDS”** με δικαιώματα **“Context.MODE_PRIVATE”** που σημαίνει ότι τα περιεχόμενα του αρχείου είναι προσπελάσιμα μόνο από την εφαρμογή που το δημιούργησε. Στη συνέχεια διαβάζουμε τα αντίστοιχα πεδία από τα στοιχεία που μας έδωσε ο χρήστης, θέτοντας ως προκαθορισμένη τιμή σε περίπτωση που ο χρήστης δεν έχει δώσει κάποια τιμή το **“null”** για τα αλφαριθμητικά και το **“0”** για τους ακέραιους αριθμούς. Έπειτα, στη γραμμή επτά ανοίγουμε μία σύνδεση προς τον εξυπηρετητή. Αυτό γίνεται με τη χρήση της μεθόδου **“Db4oClientServer.openClient(Db4oClientServer.newClientConfiguration(), ip, port, username, password)”**. Επειδή δεν διαθέτουμε κάποιο αρχείο ρυθμίσεων (Configuration file) και θέλουμε να χρησιμοποιήσουμε αυτό που προϋπάρχει στη Βάση Δεδομένων χρησιμοποιούμε το **“Db4oClientServer.newClientConfiguration()”**.

```
1. public Db4oSubClass(Context ctx) {
2.     SharedPreferences sharedPref =
   ctx.getSharedPreferences(ctx.getString(R.string.MyPREFERENCES
   ), Context.MODE_PRIVATE);
3.     String ip =
   sharedPref.getString(ctx.getString(R.string.ServerN), null);
4.     int port =
   sharedPref.getInt(ctx.getString(R.string.PortN), 0);
5.     String username =
   sharedPref.getString(ctx.getString(R.string.UserN), null);
6.     String password =
   sharedPref.getString(ctx.getString(R.string.PasswordN),
   null);
7.     db =
   Db4oClientServer.openClient(Db4oClientServer.newClientConfigu
   ration(), ip, port, username, password);
8. }
```

Εικόνα 22 “Μέθοδος Db4oSubClass”

Τέλος, με τη μέθοδο **“public void CloseDB()”**, κώδικας της Εικόνας 23, τερματίζουμε την υπάρχουσα σύνδεση. Αυτό αποτελεί βασικό στοιχείο της διαδικασίας επικοινωνίας Πελάτη Εξυπηρετητή καθώς εκτός από το γεγονός ότι αποδεσμεύονται οι πόροι που δεσμεύτηκαν για την επικοινωνία αυτή και ύστερα από έρευνα που διενεργήθηκε για την αποσφαλμάτωση ορισμένων δυσλειτουργιών κατά τη διάρκεια ανάπτυξης της εφαρμογής μας ορισμένες φορές κρίνεται αναγκαία η διακοπή και επανέναρξη της σύνδεσης με τη Βάση για την ορθή λειτουργία της εφαρμογής ακόμα και εντός της ίδιας μεθόδου.

```
1. public void CloseDB() {
2.     getDb().close();
3. }
```

Εικόνα 23 "Μέθοδος CloseDB"

3.1.2 ReflectMTypes.java

Η κλάση **"ReflectMTypes"**, κώδικας της Εικόνας 24, αποτελεί μία βοηθητική κλάση λειτουργία της οποίας είναι να συγκεντρώσει και να αναπαραστήσει με όσο δυνατόν πιο κατανοητό τρόπο τύπους μεταβλητών. Με τη χρήση αυτής της κλάσης χειριζόμαστε τους τύπους μεταβλητών ως μία αφαίρεση μέσα στις υπόλοιπες κλάσεις, για παράδειγμα σε περίπτωση που η Βάση Δεδομένων αντί να χρησιμοποιεί τον τύπο μεταβλητής "int" χρησιμοποιεί τον τύπο μεταβλητής "com.db4o.integer" τότε αρκεί μόνο να προβούμε στην αλλαγή μόνο στην κλάση "ReflectMTypes" αφήνοντας ανέγγιχτο τον υπόλοιπο κώδικα μας. Σε διαφορετική περίπτωση με τη μή ύπαρξη της κλάσης αυτής θα έπρεπε να γράψουμε σε κάθε κλάση ξεχωριστά τις συγκεκριμένες μεταβλητές και σε περίπτωση αλλαγής ο προγραμματιστής θα έπρεπε να τις αλλάξει σε κάθε κλάση ξεχωριστά.

```
1. public class ReflectMTypes {
2.
3.     public static final String STRING = "java.lang.String";
4.     public static final String SQLDATE = "java.sql.Date";
5.     public static final String UTILDATE = "java.util.Date";
6.     public static final String INT = "int";
7.     public static final String FLOAT = "float";
8.     public static final String BYTE = "byte";
9.     public static final String SHORT = "short";
10.        public static final String LONG = "long";
11.        public static final String DOUBLE = "double";
12.        public static final String BOOLEAN = "boolean";
13.        public static final String CHAR = "char";
14.    }
```

Εικόνα 24 "Κλάση ReflectMTypes"

3.1.3 Constants.java

Η κλάση **“Constants”**, κώδικας της Εικόνας 25, αποτελεί μία βοηθητική κλάση λειτουργία της οποίας είναι να συγκεντρώσει και να αναπαραστήσει με όσο δυνατόν πιο κατανοητό τρόπο τις διάφορες σταθερές που χρησιμοποιήθηκαν. Με τη χρήση αυτής της κλάσης χειριζόμαστε τις σταθερές ως μία αφαίρεση μέσα στις υπόλοιπες κλάσεις, αυτό έχει σαν αποτέλεσμα την επαναχρησιμοποίηση των σταθερών αυτών σε διάφορες κλάσεις χωρίς την ανάγκη να γνωρίζει ο προγραμματιστής την ακριβή τιμή αυτών κατά τη συγγραφή του κώδικα.

```
1. public class Constants {
2.
3.     public static final int EQUALS_OPERATOR = 0;
4.     public static final int GREATER_OPERATOR = 1;
5.     public static final int SMALLER_OPERATOR = 2;
6.     public static final int GREATER_EQUALS_OPERATOR = 3;
7.     public static final int SMALLER_EQUALS_OPERATOR = 4;
8.     public static final int LIKE_OPERATOR = 5;
9.
10.        public static final int AND_OPERATOR = 0;
11.        public static final int OR_OPERATOR = 1;
12.
13.        public static final int REQUEST_CODE = 1;
14.    }
```

Εικόνα 25 “Κλάση Constants”

3.1.4 MyConstraint.java

Η κλάση **“MyConstraint”**, κώδικας της Εικόνας 26, είναι μία κλάση POJO, δηλαδή ένα αντικείμενο Java που δεν εμπίπτει στους αυστηρούς περιορισμούς αυτής της γλώσσας προγραμματισμού. Κύρια λειτουργία αυτής της κλάσης είναι η προτυποποίηση για τη δημιουργία ενός JSON αρχείου ή μιας μεταβλητής που αποθηκεύεται το JSON περιεχόμενο, όπως συμβαίνει και στην περίπτωσή μας. Στην πρώτη γραμμή ορίζουμε ότι κατά τη σειριοποίηση των δεδομένων δεν επιθυμούμε να περιέχει το JSON αρχείο πεδία με τιμές **“null”** ή **“0”**, αν πρόκειται για αλφαριθμητικό ή αριθμό αντίστοιχα. Με βάση τα συγκεκριμένα πεδία, όπως φαίνεται στον κώδικα της Εικόνας 26 σε περίπτωση που είχαμε τις τιμές path: participates/athlete/trainer, μη ορισμένο value και operator: 1, στο παραγόμενο αρχείο θα δούμε ότι η παράμετρος value απουσιάζει. Αυτό το χαρακτηριστικό μπορεί να δηλωθεί είτε σε επίπεδο κλάσης, όπως κάναμε εμείς, είτε σε κάθε

παράμετρο / μέθοδο ξεχωριστά. Στη συνέχεια, γραμμές δύο μέχρι επτά, ορίζουμε τη σειρά με την οποία θα αναπαρασταθούν τα πεδία στο αρχείο JSON. Επειδή ορίζουμε επακριβώς με αλφαριθμητικά τα πεδία χρησιμοποιούμε ένα πίνακα τύπου αλφαριθμητικό, άλλη εναλλακτική για τη σειριοποίηση των δεδομένων είναι η αλφαβητική κατά την οποία ανάλογα με το όνομα της μεταβλητής / μεθόδου τοποθετείται το πεδίο στο αρχείο. Σε περίπτωση που δεν προβούμε στον ορισμό της σειράς των πεδίων είτε με το όνομα των πεδίων είτε αλφαβητικά, τότε τα δεδομένα τοποθετούνται σε τυχαία σειρά καθιστώντας δύσκολο έως αδύνατο το ανάγνωσμα των πεδίων από το αρχείο αυτό [18].

```
1. @JsonInclude(JsonInclude.Include.NON_NULL)
2. @JsonPropertyOrder({
3.         "path",
4.         "value",
5.         "operator",
6.         "reflectFieldType"
7. })
```

Εικόνα 26 "Κλάση MyConstraint"

Η συγκεκριμένη κλάση μπορεί να χωριστεί σε δύο επιμέρους κομμάτια. Το πρώτο είναι οι μέθοδοι ορισμού τιμής μεταβλητών (μέθοδοι set), κώδικας της Εικόνας 27, που μας επιτρέπουν να προσφέρουμε μία δημόσια διεπαφή στον χρήστη χωρίς να γνωστοποιούμε σε αυτόν τι συμβαίνει εσωτερικά της κλάσης, για παράδειγμα θα μπορούσε να γίνεται σε αυτές τις μεθόδους κάποιος έλεγχος εγκυρότητας της τιμής ή κάποια αριθμητική πράξη. Στη δικιά μας περίπτωση οριστικοποιούμε τη διαδρομή **"path"** του πεδίου στο οποίο έχουμε θέσει περιορισμό, για παράδειγμα αν είμαστε στην κλάση "Participates" και θέλουμε να ορίσουμε περιορισμό στο όνομα του εκπαιδευτή "Trainer" στη μεταβλητή **"path"** θα αποθηκευτεί η τιμή "athlete.trainer.name". Με το πεδίο **"value"** ορίζουμε την τιμή που θέλουμε να έχει το πεδίο στον περιορισμό, επειδή η τιμή μπορεί να είναι οποιοσδήποτε τύπου, αλφαριθμητικό, αριθμητικό, ημερομηνία, κλπ, αποθηκεύουμε στο JSON αρχείο ως αλφαριθμητικό έτσι ώστε να μπορέσουμε στη συνέχεια να κάνουμε εύκολα τη μετατροπή στον επιθυμητό τύπο δεδομένων. Με τη μεταβλητή **"operator"** ορίζουμε το σύμβολο που θα χρησιμοποιηθεί για σύγκριση (>,<,=,like,=<,=>), επιλέξαμε την χρήση ενός αριθμού για την διαχείριση αυτής της λειτουργίας και χρησιμοποιούμε την κλάση **"Constants"** για την μετάφραση από τον κωδικό-αριθμό στο αντίστοιχο σύμβολο. Τέλος, με τη μεταβλητή **"reflectFieldtype"** ορίζουμε τον τύπο αυτής, αλφαριθμητικό, αριθμητικό, ημερομηνία κλπ και τη χρησιμοποιούμε ως επί το πλήστον για τη σωστή μετατροπή του πεδίου **"value"** στον σωστό τύπο, όπως αναφέρθηκε παραπάνω.

```
1. @JsonProperty("reflectFieldType")
2. public void setReflectFieldType(String reflectFieldType) {
3.     this.reflectFieldType = reflectFieldType;
4. }
5.
6. @JsonProperty("operator")
7. public void setOperator(Integer operator) {
8.     this.operator = operator;
9. }
10.
11.    @JsonProperty("value")
12.    public void setValue(String value) {
13.        this.value = value;
14.    }
15.
16.    @JsonProperty("path")
17.    public void setPath(List<String> path) {
18.        this.path = path;
19.    }
```

Εικόνα 27 "Μέθοδοι Set"

Το δεύτερο κομμάτι περιλαμβάνει τις μεθόδους προσπέλασης τιμών (μέθοδοι Get), κώδικας της Εικόνας 28, οι οποίες μας επιτρέπουν να προσφέρουμε μία δημόσια διεπαφή στον χρήστη χωρίς να γνωστοποιούμε σε αυτόν τι συμβαίνει εσωτερικά της κλάσης για την επιστροφή των τιμών των πεδίων που αναφέραμε παραπάνω. Αυτό μας επιτρέπει να έχουμε ως ένα βαθμό ασφάλεια στην ακεραιότητα των τιμών των δεδομένων, καθώς απαγορεύουμε στον χρήστη να έχει άμεση επαφή με την εκάστοτε μεταβλητή.

```
1. @JsonProperty("reflectFieldType")
2. public String getReflectFieldType() {
3.     return reflectFieldType;
4. }
5.
6. @JsonProperty("operator")
7. public Integer getOperator() {
8.     return operator;
9. }
10.
11.    @JsonProperty("value")
12.    public String getValue() {
13.        return value;
14.    }
15.
16.    @JsonProperty("path")
17.    public List<String> getPath() {
18.        return path;
19.    }
```

Εικόνα 28 "Μέθοδοι Get"

Τέλος,στον κώδικα της Εικόνας 29, βλέπουμε τις μεταβλητές της κλάσης που αναφέραμε παραπάνω. Οι μεταβλητές είναι δηλωμένες ως ιδιωτικές **"private"** καθώς δεν θέλουμε να είναι προσπελάσιμες εκτός της συγκεκριμένης κλάσης. Αξίζει να αναφέρουμε ότι η γραμμή πάνω από κάθε μεταβλητή είναι για την ονοματοδοσία αυτών εσωτερικά του αρχείου JSON. Αν δεν είχαμε προβεί στην ενέργεια αυτή τότε μέσα στο αρχείο θα είχαμε το όνομα της κάθε μεθόδου – μεταβλητής και όχι μία κοινή ονομασία για κάθε μεταβλητή, σε άλλη περίπτωση θα μπορούσε ωστόσο το όνομα της μεταβλητής της κλάσης να είναι διαφορετικό από αυτό του JSON αρχείου.

```
1. @JsonProperty("path")
2. private List<String> path = new ArrayList<String>();
3. @JsonProperty("value")
4. private String value;
5. @JsonProperty("operator")
6. private Integer operator;
7. @JsonProperty("reflectFieldType")
8. private String reflectFieldType;
```

Εικόνα 29 "Οι Μεταβλητές της Κλάσης MyConstraint"

3.1.5 ConstraintsJsonData.java

Η κλάση **"ConstraintsJsonData"** είναι παρόμοια με την **"MyConstraint"** καθώς και οι δύο αφορούν την διαχείριση αρχείου τύπου JSON. Η χρήση της κλάσης αυτής είναι πάρα πολύ σημαντική καθώς μας επιτρέπει να αποθηκεύσουμε μία σειρά από περιορισμούς, τύπου **"MyConstraint"** στο JSON αρχείο, καθώς ο χρήστης δύναται να ορίσει περισσότερους από έναν περιορισμούς. Δυστυχώς, δίχως τη βοηθητική κλάση αυτή δεν θα ήταν δυνατό να αποθηκεύσουμε πέραν του ενός περιορισμού στο JSON αρχείο. Εκτός από τη λίστα από περιορισμούς που υπάρχει στην κλάση αυτή έχουμε και το πεδίο **"operator"**, το οποίο χρησιμοποιείται για την ένωση των περιορισμών είτε με την χρήση λογικού "και" είτε με τη χρήση λογικού "ή". Όπως και στην κλάση **"MyConstraint"**, έτσι και σε αυτή για τη μεταβλητή operator χρησιμοποιούμε αριθμό τον οποίο και έχουμε δηλώσει στην κλάση **"Constants"**. Αξίζει να αναφερθεί, κώδικας της Εικόνας 30, ότι σε αντίθεση με την κλάση **"MyConstraint"** όπου δηλώνουμε τη σειρά όλων των πεδίων, στην **"ConstraintsJsonData"** δηλώνουμε μόνο τη λίστα **"constraints"**, αυτό έχει σαν αποτέλεσμα να τοποθετούνται οι διάφορες λίστες περιορισμών, στις περιπτώσεις που είναι περισσότερες από μία, και στο τέλος να τοποθετείται η μεταβλητή **"operator"** της κλάσης **"ConstraintsJsonData"**.

```
1. @JsonInclude(JsonInclude.Include.NON_NULL)
2. @JsonPropertyOrder({
3.         "constraints"
4. })
```

Εικόνα 30 "Κλάση ConstraintsJsonData"

3.1.6 DividerItemDecoration.java

Η κλάση **“DividerItemDecoration”**, κώδικας της Εικόνας 31, κληρονομεί την κλάση **“RecyclerView.ItemDecoration”** που αποτελεί ένα εργαλείο για τη διακόσμηση των αντικειμένων παιδιών ενός αντικειμένου RecyclerView. Αυτό αποτελεί ένα άκρως χρήσιμο χαρακτηριστικό καθώς μας δίνει τη δυνατότητα να μετασχηματίσουμε τη διακόσμηση ενός αντικειμένου παιδιού του RecyclerView, να δημιουργήσουμε δικά μας γραφικά στοιχεία για κάθε αντικείμενο, να δημιουργήσουμε με τη χρήση γραφικών απεικονίσεων λογικές ομάδες αντικειμένων εντός του αντικειμένου RecyclerView [19] καθώς και να καθορίσουμε το offset του κάθε αντικειμένου παιδιού του αντικειμένου RecyclerView. Στην αρχή της κλάσης **“DividerItemDecoration”** δηλώνουμε τις μεταβλητές **“ATTRS”**, τύπου πίνακας ακεραίων αριθμών, και **“mDivider”** τύπου **“Drawable”**. Στην πρώτη περίπτωση αποθηκεύουμε το “id” για το στοιχείο “listDivider”, εντός του καταλόγου drawable, όπως αυτό καθορίζεται από το Λειτουργικό Σύστημα Android και έχει σταθερή τιμή ίση με 16843284 [20]. Στη δεύτερη περίπτωση, θα αποθηκεύσουμε τον διαχωριστή (divider) που θα περάσουμε παραμετρικά μέσω του δομητή και θα σχεδιαστεί ανάμεσα στα αντικείμενα παιδιά του αντικειμένου RecyclerView. Η μέθοδος **“public DividerItemDecoration(Context context)”** αποτελεί τον δομητή της κλάσης μας και δέχεται ως είσοδο ένα αντικείμενο τύπου **Context** που στην περίπτωσή μας είναι η αναφορά στο αντικείμενο RecycleView. Στη γραμμή δύο, η μεταβλητή **“styledAttributes”** τύπου **“TypedArray”**, τύπος πίνακα που αποθηκεύει αντικείμενα που ανασύρονται από τη χρήση της μεθόδου **“obtainStyledAttributes()”**, στη συνέχεια περνώντας παραμετρικά τον πίνακα **“ATTRS”** στην μέθοδο **“obtainStyledAttributes()”** λαμβάνουμε έναν πίνακα τύπου **“TypedArray”** με τιμές πεδίων που αφορούν το Γραφικό Περιβάλλον και πιο συγκεκριμένα την θέση του κάθε στοιχείου στον χώρο της οθόνης. Στη γραμμή τρία με τη χρήση της μεθόδου **“getDrawable()”** λαμβάνουμε το αντικείμενο τύπου **Drawable** που βρίσκεται στην πρώτη θέση του πίνακα τύπου **“TypedArray”** που ορίσαμε νωρίτερα. Σε περίπτωση που θέλουμε να προσπελάσουμε αντικείμενο που δεν είναι τύπου “color” ή “drawable” η μέθοδος **“getDrawable()”** επιστρέφει Exception. Τέλος, στη γραμμή τέσσερα καλούμε τη μέθοδο **“recycle()”**, πράγμα που κρίνεται επιβεβλημένο μετά τη χρήση κάποιου αντικειμένου τύπου “TypedArray”.

```
1. public class DividerItemDecoration extends
   RecyclerView.ItemDecoration {
2.
3.    private static final int[] ATTRS = new
   int[]{android.R.attr.listDivider};
4.
5.    private Drawable mDivider;
6.
7.    /**
8.     * Default divider will be used
9.     */
10.      public DividerItemDecoration(Context context) {
11.          final TypedArray styledAttributes =
   context.obtainStyledAttributes(ATTRS);
12.          mDivider = styledAttributes.getDrawable(0);
13.          styledAttributes.recycle();
14.      }
```

Εικόνα 31 "Κλάση DividerItemDecoration"

Η μέθοδος **"public void onDraw(Canvas c, RecyclerView parent, RecyclerView.State state)"**, κώδικας της Εικόνας 32, η οποία καλείται μόνο μία φορά, είναι επιφορτισμένη με τον καθορισμό των διαστάσεων και τη σχεδίαση του διαχωριστή (divider) στο αντικείμενο RecyclerView. Στις γραμμές δύο και τρία υπολογίζουμε τα όρια δεξιά και αριστερά του divider με βάση τα αντίστοιχα όρια του αντικειμένου τύπου RecyclerView. Στη συνέχεια για κάθε αντικείμενο παιδί του αντικειμένου RecyclerView υπολογίζουμε τα άνω και κάτω όρια, καθώς είναι μοναδικά για κάθε αντικείμενο παιδί. Στη γραμμή δεκατέσσερα, καθορίζουμε τις διαστάσεις του εκάστοτε divider και στη γραμμή δεκαπέντε σχεδιάζουμε το αντικείμενο.

```
1. public void onDraw(Canvas c, RecyclerView parent,
   RecyclerView.State state) {
2.     int left = parent.getPaddingLeft();
3.     int right = parent.getWidth() - parent.getPaddingRight();
4.
5.     int childCount = parent.getChildCount();
6.     for (int i = 0; i < childCount; i++) {
7.         View child = parent.getChildAt(i);
8.
9.         RecyclerView.LayoutParams params =
   (RecyclerView.LayoutParams) child.getLayoutParams();
10.
11.          int top = child.getBottom() + params.bottomMargin;
12.          int bottom = top + mDivider.getIntrinsicHeight();
13.
14.          mDivider.setBounds(left, top, right, bottom);
15.          mDivider.draw(c);
16.      }
17.  }
```

Εικόνα 32 "Μέθοδος onDraw"

3.1.7 ReflectClassesResultsRecyclerViewAdapter.java

Το RecyclerView αποτελεί μία αναβαθμισμένη έκδοση του αντικειμένου ListView και μπορεί να χρησιμοποιηθεί από συσκευές με έκδοση του Λειτουργικού Συστήματος Android προγενέστερες της έκδοσης 5.0.0 με τη χρήση της βιβλιοθήκης support-v7. Ίσως το σημαντικότερο κομμάτι του αντικειμένου RecyclerView είναι η κλάση RecyclerView.Adapter. Με τη χρήση αυτής της κλάσης ενώνουμε τα δεδομένα που λαμβάνουμε από τη Βάση Δεδομένων με το Γραφικό Περιβάλλον. Στη συγκεκριμένη περίπτωση, κώδικας της Εικόνας 33, ορίζουμε αρχικά τις global μεταβλητές **"mValues"** τύπου λίστας αλφαριθμητικό, όπου θα αποθηκεύσουμε τις τιμές που στη συνέχεια θα προβάλλουμε. Επειδή επιλέξαμε να υλοποιήσουμε τη λειτουργικότητα του γεγονότος "OnItemClickListener" στην κλάση **RecursivePrint**, δημιουργούμε έναν listener που θα μας ενημερώνει πότε τελείωσε η διαδικασία που ορίσαμε στην κλάση **ReflectClassesResultsRecyclerViewAdapter**. Στη συνέχεια δημιουργούμε τον δομητή της κλάσης **"public ReflectClassesResultsRecyclerViewAdapter(List<String> items, RecursivePrint.OnReflectClassItemClickedListener listener)"**, όπου και ορίζουμε τις τιμές των global μεταβλητών μας. Η μέθοδος **"public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType)"**, επιστρέφει το View που θα εμφανίζεται, για κάθε View θα πρέπει να δημιουργήσουμε ένα νέο

αντικείμενο τύπου **ViewHolder**. Η μέθοδος αυτή καλείται κάθε φορά που θέλουμε να δημιουργήσουμε ένα νέο instance.

```
1. public class ReflectClassesResultsRecyclerViewAdapter extends
   RecyclerView.Adapter<ReflectClassesResultsRecyclerViewAdapter
   .ViewHolder> {
2.
3.    private final List<String> mValues;
4.    private final
   RecursivePrint.OnReflectClassItemClickedListener mListener;
5.
6.    public
   ReflectClassesResultsRecyclerViewAdapter(List<String> items,
   RecursivePrint.OnReflectClassItemClickedListener listener) {
7.        this.mValues = items;
8.        mListener = listener;
9.    }
10.
11.     @Override
12.     public ViewHolder onCreateViewHolder(ViewGroup parent,
   int viewType) {
13.         View view =
   LayoutInflater.from(parent.getContext())
14.                 .inflate(R.layout.recursive_field_item,
   parent, false);
15.         return new ViewHolder(view);
16.     }
```

Εικόνα 33 " Κλάση ReflectClassesResultRecyclerViewAdapter"

Στη συνέχεια, κώδικας της Εικόνας 34, η μέθοδος **"public void onBindViewHolder(ViewHolder holder, final int position)"**, δέχεται ως είσοδο ένα αντικείμενο τύπου **ViewHolder** καθώς και την θέση που βρίσκεται το επιλεγμένο αντικείμενο. Η συγκεκριμένη μέθοδος καλείται κάθε φορά που παρουσιάζουμε αντικείμενα στο Γραφικό Περιβάλλον. Αρχικά, στη γραμμή δύο αποθηκεύουμε προσωρινά την τιμή του επιλεγμένου αντικειμένου, αντικείμενο που βρίσκεται στη θέση **"position"** της λίστας αλφαριθμητικών **"mValues"**. Στη συνέχεια, γραμμή τρία αποθηκεύουμε το επιλεγμένο πεδίο και στη γραμμή τέσσερα το προβάλλουμε στο χρήστη. Στη συνέχεια, όταν τελειώσει η διαδικασία επιστρέφεται η θέση του επιλεγμένου αντικειμένου. Η τρίτη μέθοδος που κληρονομείται από την κλάση RecyclerView.Adapter είναι η **"public int getItemCount()"**, η οποία απλά επιστρέφει τον αριθμό των αντικειμένων από τα οποία ο χρήστης μπορεί να επιλέξει.

```
1. @Override
2. public void onBindViewHolder(ViewHolder holder, final int
   position) {
3.     String textToShow = mValues.get(position);
4.     holder.mItem = textToShow;//deixnei to epilegmeno
   antikeimeno
5.     holder.mNameView.setText(textToShow);//probalei to
   antikeimeno pou 8es na emfaniseis
6.
7.     holder.mView.setOnClickListener(new View.OnClickListener()
   {
8.         @Override
9.         public void onClick(View v) {
10.             if (null != mListener) {
11.                 // Notify the active callbacks interface
   (the activity, if the
12.                 // fragment is attached to one) that an
   item has been selected.
13.                 mListener.onListItemClicked(position);
14.             }
15.         }
16.     });
17. }
18.
19. @Override
20. public int getItemCount() {
21.     return mValues.size();
22. }
```

Εικόνα 34 "Μέθοδοι onBindViewHolder και getItemCount"

Η κλάση **ViewHolder**, κώδικας της Εικόνας 35, αποτελεί μια cache των τιμών που θα εμφανιστούν στο RecyclerView, στη συγκεκριμένη κλάση γίνεται η σύνδεση του Γραφικού Περιβάλλοντος με τα δεδομένα. Στις γραμμές δύο έως τέσσερα δηλώνουμε τις μεταβλητές που θα αποθηκεύουν το πεδίο που έχουμε επιλέξει καθώς και το αντικείμενο τύπου View που θα διαχειριζόμαστε. Η μέθοδος **"public ViewHolder(View view)"** αποτελεί τον δομητή της κλάσης, όπου ορίζουμε τις τιμές των πεδίων. Τέλος, η μέθοδος **"public String toString() "** που κληρονομείται από την κλάση RecyclerView.ViewHolder χρησιμοποιείται για την εκτύπωση πληροφοριών που είναι ιδιαιτέρως χρήσιμες κατά τη διάρκεια της αποσφαλμάτωσης.

```
1. public class ViewHolder extends RecyclerView.ViewHolder {
2.     public final View mView;
3.     public final TextView mNameView;
4.     public String mItem;
5.
6.     public ViewHolder(View view) {
7.         super(view);
8.         mView = view;
9.         mNameView = (TextView)
   view.findViewById(R.id.itemName);
10.      }
11.
12.      @Override
13.      public String toString() {
14.          return super.toString();
15.      }
16.  }
```

Εικόνα 35 "Κλάση ViewHolder"

3.1.8 ReflectFieldsRecyclerViewAdapter.java

Με τη χρήση της κλάσης **"ReflectFieldsRecyclerViewAdapter"** ενώνουμε τα δεδομένα που λαμβάνουμε από τη Βάση Δεδομένων με το Γραφικό Περιβάλλον και πιο συγκεκριμένα τα ονόματα των πεδίων της κάθε κλάσης με το αντικείμενο RecyclerView της Activity **ConstraintsActivity**. Στον κώδικα της Εικόνας 36, ορίζουμε αρχικά τις global μεταβλητές **"mValues"** τύπου λίστας αντικειμένου **ReflectField**, όπου θα αποθηκεύσουμε τις τιμές που στη συνέχεια θα προβάλλουμε. Για να γνωρίζουμε σε ποιά πεδία έχουμε θέσει περιορισμούς ορίζουμε μία μεταβλητή τύπου πίνακα Boolean όπου θα ορίζουμε ως αληθές κάθε πεδίο που του έχει ανατεθεί κάποιος περιορισμός. Επειδή επιλέξαμε να υλοποιήσουμε τη λειτουργικότητα του γεγονότος OnItemLongClickListener στην κλάση **ConstraintsActivity**, δημιουργούμε έναν listener που θα μας ενημερώνει πότε τελείωσε η διαδικασία που ορίσαμε στην κλάση **ReflectFieldsRecyclerViewAdapter**. Στη συνέχεια δημιουργούμε τον δομητή της κλάσης **"public ReflectFieldsRecyclerViewAdapter(List<ReflectField> items, ConstraintsActivity.OnListItemLongClickedListener listener)"**, όπου και ορίζουμε τις τιμές των global μεταβλητών μας. Η μέθοδος **"public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType)"**, επιστρέφει το View που θα εμφανίζεται, για κάθε View θα πρέπει να δημιουργήσουμε ένα νέο αντικείμενο τύπου **ViewHolder**. Η μέθοδος αυτή καλείται κάθε φορά που θέλουμε να δημιουργήσουμε ένα νέο instance.

```
1. public class ReflectFieldsRecyclerViewAdapter extends
   RecyclerView.Adapter<ReflectFieldsRecyclerViewAdapter.ViewHol
   der> {
2.
3.    private final List<ReflectField> mValues;
4.    private boolean[] hasConstraint;
5.    private final
   ConstraintsActivity.OnListItemLongClickedListener mListener;
6.
7.    public ReflectFieldsRecyclerViewAdapter(List<ReflectField>
   items, ConstraintsActivity.OnListItemLongClickedListener
   listener) {
8.        mValues = items;
9.        mListener = listener;
10.          hasConstraint = new boolean[mValues.size()];
11.      }
12.
13.      @Override
14.      public ViewHolder onCreateViewHolder(ViewGroup parent,
   int viewType) {
15.          View view =
   LayoutInflater.from(parent.getContext())
16.                  .inflate(R.layout.reflect_field_item,
   parent, false);
17.          return new ViewHolder(view);
18.      }
```

Εικόνα 36 "Κλάση ReflectFieldsRecyclerViewAdapter"

Στη συνέχεια, κώδικας της Εικόνας 37, η μέθοδος **"public void onBindViewHolder(ViewHolder holder, final int position)"**, δέχεται ως είσοδο ένα αντικείμενο τύπου **ViewHolder** καθώς και την θέση που βρίσκεται το επιλεγμένο αντικείμενο. Αρχικά, στη γραμμή δύο αποθηκεύουμε προσωρινά την τιμή του επιλεγμένου αντικειμένου, αντικείμενο που βρίσκεται στη θέση **"position"** της λίστας τύπου αντικειμένου **ReflectFields**. Στη συνέχεια, γραμμή τρία αποθηκεύουμε το επιλεγμένο πεδίο και στη γραμμή τέσσερα το προβάλλουμε στο χρήστη. Στις γραμμές τέσσερα έως εννέα, ελέγχουμε τον τύπο δεδομένων της εκάστοτε μεταβλητής, στην περίπτωση που είναι τύπου **"Collection"**, εμφανίζουμε το όνομα της μεταβλητής και το γεγονός ότι είναι τύπου "Collection", σε αντίθετη περίπτωση εμφανίζουμε το όνομα της μεταβλητής και τον τύπο δεδομένου της. Στη συνέχεια, όταν τελειώσει η διαδικασία επιστρέφεται η θέση του επιλεγμένου αντικειμένου. Στο τέλος της μεθόδου αυτής ελέγχουμε αν υπάρχει κάποιος περιορισμός για το συγκεκριμένο πεδίο ορίζουμε το χρώμα παρασκηνίου ως κίτρινο και τη γραμματοσειρά μαύρη.

```
1. public void onBindViewHolder(ViewHolder holder, int position)
   {
2.     final ReflectField reflectField = mValues.get(position);
3.     holder.mItem = reflectField;
4.     String textToShow;
5.     if (reflectField.getFieldType().isCollection()) {
6.         textToShow = reflectField.getName() + " : " +
   "Collection";
7.     } else {
8.         textToShow = reflectField.getName() + " : " +
   reflectField.getFieldType().getName();
9.     }
10.       holder.mNameView.setText(textToShow);
11.            holder.mView.setOnLongClickListener(new
   View.OnLongClickListener() {
12.           @Override
13.           public boolean onLongClick(View v) {
14.               if (null != mListener) {
15.                mListener.onListItemLongClicked(reflectField);
16.               }
17.               return true;
18.           }
19.       });
20.       if (hasConstraint[position]) {
21.           holder.mView.setBackgroundColor(Color.YELLOW);
22.           holder.mNameView.setTextColor(Color.BLACK);
23.       } else {
24.           holder.mView.setBackgroundColor(Color.TRANSPARENT);
25.           holder.mNameView.setTextColor(Color.WHITE);
26.   }}
```

Εικόνα 37 "Μέθοδος onBindViewHolder"

Η μέθοδος **"public void setHasConstraint(ReflectField reflectField, boolean hasConstraint)"**, κώδικας της Εικόνας 38, δέχεται ως είσοδο ένα πεδίο τύπου **ReflectField** και μία λογική μεταβλητή τύπου Boolean και ενημερώνει τον πίνακα τύπου Boolean ότι στην ίδια θέση με αυτή που βρίσκεται το πεδίο στη λίστα **"mValues"** έχει οριστεί περιορισμός.

```
1. @Override
2.     public int getItemCount() {
3.         return mValues.size();
4.     }
5.
6.     public void setHasConstraint(ReflectField reflectField,
   boolean hasConstraint) {
7.         this.hasConstraint[mValues.indexOf(reflectField)] =
   hasConstraint;
8.     }
```

Εικόνα 38 "Μέθοδοι getItemCount και setHasConstraint"

3.1.9 ReflectFieldsValuesRecyclerViewAdapter.java

Με τη χρήση της κλάσης **"ReflectFieldsValuesRecyclerViewAdapter"** φορτώνουμε τα αποτελέσματα του query στο αντικείμενο RecyclerView της Activity **RecursivePrint**. Στον κώδικα της Εικόνας 39, ορίζουμε αρχικά τις global μεταβλητές **"reflectFields"** και **"mValues"** τύπου λίστας αντικειμένου **ReflectField** και λίστας αλφαριθμητικό αντίστοιχα. Επειδή επιλέξαμε να υλοποιήσουμε τη λειτουργικότητα του γεγονότος OnItemClickListener στην κλάση **RecursivePrint**, δημιουργούμε έναν listener που θα μας ενημερώνει πότε τελείωσε η διαδικασία που ορίσαμε στην κλάση **ReflectFieldsValuesRecyclerViewAdapter**. Στη συνέχεια δημιουργούμε τον δομητή της κλάσης **"public ReflectFieldsValuesRecyclerViewAdapter(List<String> items, List<ReflectField> reflectFields, RecursivePrint.OnReflectFieldItemClickedListener listener)"**, όπου και ορίζουμε τις τιμές των global μεταβλητών μας. Η μέθοδος **"public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType)"**, επιστρέφει το View που θα εμφανίζεται, για κάθε View θα πρέπει να δημιουργήσουμε ένα νέο αντικείμενο τύπου **ViewHolder**. Η μέθοδος αυτή καλείται κάθε φορά που θέλουμε να δημιουργήσουμε ένα νέο instance.

```
1. public class ReflectFieldsValuesRecyclerViewAdapter extends
   RecyclerView.Adapter<ReflectFieldsValuesRecyclerViewAdapter.V
   iewHolder> {
2.     private final List<ReflectField> reflectFields;
3.     private final List<String> mValues;
4.     private final
   RecursivePrint.OnReflectFieldItemClickedListener mListener;
5.
6.     public ReflectFieldsValuesRecyclerViewAdapter(List<String>
   items, List<ReflectField> reflectFields,
   RecursivePrint.OnReflectFieldItemClickedListener listener) {
7.         this.reflectFields = reflectFields;
8.         this.mValues = items;
9.         mListener = listener;
10.      }
11.
12.      @Override
13.      public ViewHolder onCreateViewHolder(ViewGroup parent,
   int viewType) {
14.          View view =
   LayoutInflater.from(parent.getContext())
15.                  .inflate(R.layout.recursive_field_item,
   parent, false);
16.          return new ViewHolder(view);
17.      }
```

Εικόνα 39 "Κλάση ReflectFieldsValuesRecyclerViewAdapter"

Στη συνέχεια, κώδικας της Εικόνας 40, η μέθοδος **"public void onBindViewHolder(ViewHolder holder, final int position)"**, δέχεται ως είσοδο ένα αντικείμενο τύπου **ViewHolder** καθώς και την θέση που βρίσκεται το επιλεγμένο αντικείμενο. Η συγκεκριμένη μέθοδος καλείται κάθε φορά που παρουσιάζουμε αντικείμενα στο Γραφικό Περιβάλλον. Αρχικά, στη γραμμή δύο αποθηκεύουμε προσωρινά την τιμή του επιλεγμένου αντικειμένου, αντικείμενο που βρίσκεται στη θέση **"position"** της λίστας αντικειμένων τύπου **ReflectFields**.Μετά, στη γραμμή τρία αποθηκεύουμε προσωρινά την τιμή του επιλεγμένου αντικειμένου, αντικείμενο που βρίσκεται στη θέση **"position"** της λίστας αλφαριθμητικών **"mValues"**. Στη συνέχεια, γραμμή τέσσερα αποθηκεύουμε το επιλεγμένο πεδίο και στη γραμμή πέντε δημιουργούμε ένα αλφαριθμητικό με το όνομα του πεδίου καθώς και την τιμή αυτού. Στη γραμμή έξι εμφανίζουμε στο RecyclerView το παραπάνω πεδίο. Στη συνέχεια, όταν τελειώσει η διαδικασία επιστρέφεται η θέση του επιλεγμένου αντικειμένου. Η τρίτη μέθοδος που κληρονομείται από την κλάση RecyclerView.Adapter είναι η **"public int getItemCount()"**, η οποία απλά επιστρέφει τον αριθμό των αντικειμένων από τα οποία ο χρήστης μπορεί να επιλέξει.

```
1. @Override
2. public void onBindViewHolder(ViewHolder holder, int position)
   {
3.     final ReflectField reflectField =
   reflectFields.get(position);
4.     final String value = mValues.get(position);
5.     holder.mItem = value;
6.     String textToShow = reflectField.getName() + " : " +
   value;
7.     holder.mNameView.setText(textToShow);
8.     holder.mView.setOnClickListener(new View.OnClickListener()
   {
9.         @Override
10.          public void onClick(View v) {
11.              if (null != mListener) {
12.                  mListener.onListItemClicked(value,
   reflectField);
13.              }
14.          }
15.      });
16.  }
17.  @Override
18.  public int getItemCount() {
19.      return mValues.size();
20.  }
```

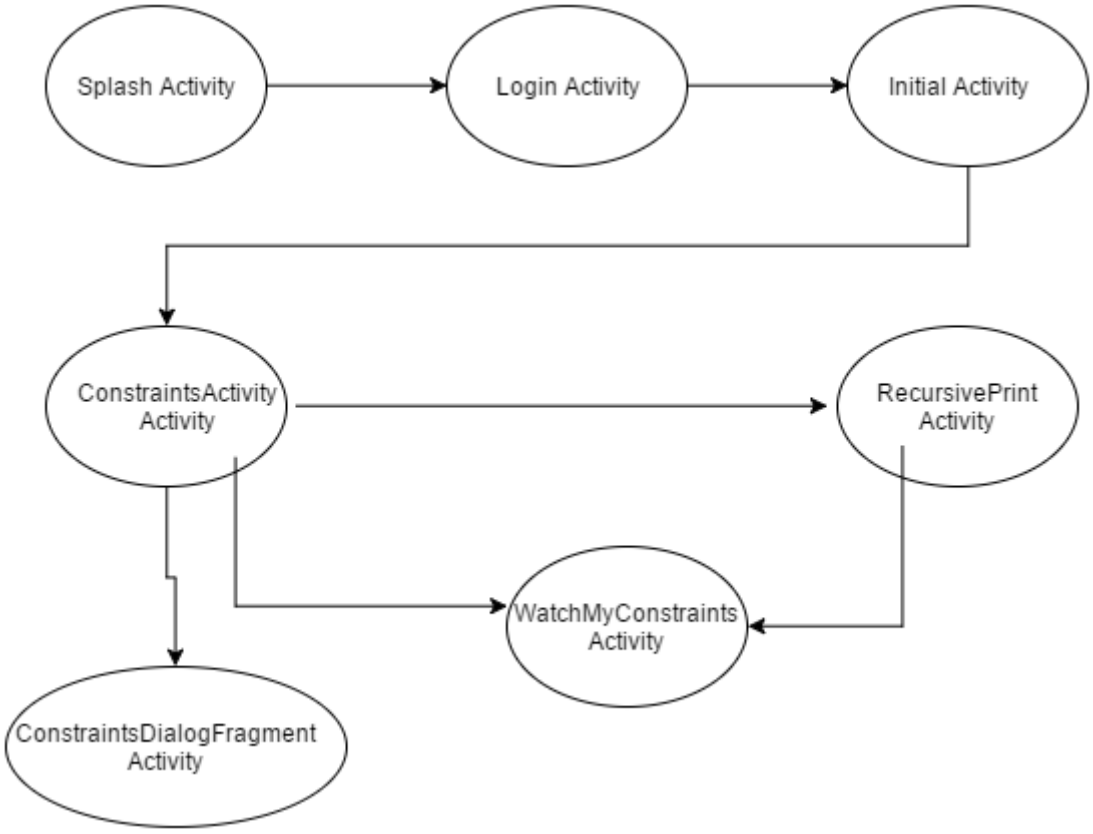
Εικόνα 40 "Μέθοδος onBindViewHolder"

3.2 Κλάσεις που κληρονομούν την κλάση Activity

Η κλάση Activity επιφορτίζεται με τη δημιουργία ενός παραθυρικού περιβάλλοντος και την τοποθέτηση των διαφόρων στοιχείων (Views) σε αυτό. Κάθε κλάση που την κληρονομεί είναι υπεύθυνη για την διαχείριση ενός Γραφικού Περιβάλλοντος και είναι ανεξάρτητη από κάθε άλλη κλάση που κληρονομεί την κλάση Activity. Ωστώσο, σημαντικό κομμάτι μιας εφαρμογής είναι η σύνδεση των κλάσεων αυτών έτσι ώστε να ενισχύσουμε την εμπειρία χρήστη. Η εφαρμογή μας βασίζεται σε επτά κλάσεις που κληρονομούν την κλάση Activity. Η σειρά με την οποία χρησιμοποιούνται φαίνεται στην Εικόνα 41 και η λειτουργίες αυτών είναι οι κάτωθι:

Splash Activity: Αρχική Activity που περιέχει το λογότυπο της εφαρμογής.
Login Activity: Περιέχει τη φόρμα συμπλήρωσης των στοιχείων για σύνδεση με τη Βάση Δεδομένων.
Initial Activity: Ο χρήστης επιλέγει την κλάση ρίζα για την δημιουργία query και βλέπει τα πεδία της κλάσης.
ConstraintsActivity Activity: Ο χρήστης επιλέγει τα πεδία στα οποία θα θέσει περιορισμούς.

ConstraintsDialogFragment: Το Παραθυρικό Περιβάλλον όπου ο χρήστης ορίζει τον τελεστή και την τιμή του περιορισμού.
WatchMyConstraints Activity: Σε αυτή την Activity ο χρήστης μπορεί να δει τους περιορισμούς που έχει θέσει.
RecursivePrint Activity: Ο χρήστης προβάλλει τα αποτελέσματα του query που εκτέλεσε.



Εικόνα 41 "Διάγραμμα κλάσεων που κληρονομούν την Activity"

3.2.1 Splash.java

Η Activity **Splash**, κώδικας της Εικόνας 42, είναι μία απλή Activity που κληρονομεί την κλάση **AppCompatActivity** η οποία μας δίνει τη δυνατότητα να χρησιμοποιήσουμε τις αλλαγές που προώθησε η Google Inc. μέσω της έκδοσης 5.0 του Λειτουργικού Συστήματος Android τόσο στο Γραφικό Περιβάλλον όσο και στον τρόπο που διαχειρίζεται το Λειτουργικό Σύστημα τα γεγονότα, από την έκδοση 2.1 και μετέπειτα. Η μόνη μέθοδος που υλοποιείται στην κλάση αυτή είναι η **"protected void onCreate(Bundle savedInstanceState)"**. Στην οποία αφού ορίσουμε το Γραφικό Περιβάλλον που θα χρησιμοποιήσουμε δηλώνουμε ένα συντελεστή ο οποίος στη συνέχεια θα πολλαπλασιαστεί για να δηλώσει το χρονικό διάστημα που θα βρίσκεται το Activity στο προσκήνιο, συνήθης τακτική αποτελεί να ορίζουμε μία μεταβλητή η οποία θα δηλώνει τα δευτερόλεπτα που επιθυμούμε

και στη συνέχει να την πολλαπλασιάζουμε με milliseconds (1000), καθώς οι περισσότερες μέθοδοι απαιτούν ο χρόνος να βρίσκεται στη μονάδα μέτρησης milliseconds. Με τη χρήση του αντικειμένου τύπου **Handler**, που μας δίνει τη δυνατότητα να διαχειριστούμε αντικείμενα τύπου Runnable. Το αντικείμενο τύπου Handler χρησιμοποιείται κατά κόρον, όπως και στη συγκεκριμένη περίπτωση, για την εκτέλεση κάποιας ενέργειας ύστερα από κάποιο χρονικό διάστημα. Στην μέθοδο **"onPostDelay()"** περνάμε παραμετρικά το αντικείμενο τύπου **Runnable** που θα εκτελεστεί μόλις λήξει το χρονικό διάστημα που αποτελεί τη δεύτερη παράμετρο εισόδου της μεθόδου. Μόλις λήξει το χρονόμετρο που δηλώσαμε παραπάνω τότε θα εκτελεστεί το αντικείμενο τύπου **Runnable** που στη συγκεκριμένη περίπτωση είναι η μετάβαση στην επόμενη Activity.

```
1. public class Splash extends AppCompatActivity {
2.     @Override
3.     protected void onCreate(Bundle savedInstanceState) {
4.         super.onCreate(savedInstanceState);
5.         setContentView(R.layout.activity_splash);
6.         int delay = 3;
7.         new Handler().postDelayed(new Runnable() {
8.             public void run() {
9.                 startActivity(new Intent(Splash.this,
   LoginActivity.class));
10.                 finish();
11.             }
12.         }, delay * 1000);
13.     }
14. }
```

Εικόνα 42 "Κλάση Splash"

3.2.2 LoginActivity.java

Η πρώτη Activity της εφαρμογής που εκκινείται και με την οποία αλληλεπιδρά ο χρήστης είναι η **LoginActivity** και βασίζεται στην LoginActivity που προσφέρεται από το Λειτουργικό Σύστημα Android. Στον κώδικα της Εικόνας 43, στις γραμμές δύο έως εννέα δηλώνουμε ένα αντικείμενο τύπου **UserLoginTask**, που είναι υπεύθυνο για τη διεκπαιρέωση δικτυακών λειτουργιών, και αντικείμενα που αφορούν το Γραφικό Περιβάλλον και θα τα χρησιμοποιήσουμε για να ελέγχουμε τα αντίστοιχα Components από τη διάδραση με τον χρήστη. Πιο συγκεκριμένα έχουμε τρία AutoCompleteTextView που χρησιμοποιούνται ως είσοδος για το όνομα χρήστη, την διεύθυνση στην οποία βρίσκεται ο Εξυπηρετητής και τέλος η πόρτα στην οποία ο Εξυπηρετητής αναμένει ερωτήματα για τον Εξυπηρετητή db4o. Για τον κωδικό χρήστη επιλέξαμε να χρησιμοποιήσουμε αντικείμενο τύπου EditText

που διαθέτει επιλογή κρυφό για την ασφαλή εισαγωγή κωδικού. Τα δύο αντικείμενα τύπου View μας δίνουν τη δυνατότητα να διαχειριζόμαστε τόσο το ProgressBar, για να ενημερώνουμε τον χρήστη για την πορεία της σύνδεσης με τον εξυπηρετητή, όσο και ολόκληρη τη φόρμα, γραφικό περιβάλλον, με τη χρήση των μεταβλητών **“mProgressView”** και **“mLoginFormView”** αντίστοιχα. Τέλος, το αντικείμενο τύπου Button το χρησιμοποιούμε για να προσφέρουμε στον χρήστη ένα σύντομο οδηγό χρήσης της εφαρμογής.

```
1. public class LoginActivity extends AppCompatActivity
   implements LoaderManager.LoaderCallbacks<Cursor> {
2.
3.     private UserLoginTask mAuthTask = null;
4.     private AutoCompleteTextView UsernameView;
5.     private AutoCompleteTextView UrlView;
6.     private AutoCompleteTextView PortView;
7.     private EditText PasswordView;
8.     private View mProgressView;
9.     private View mLoginFormView;
10.    private Button FAQ;
```

Εικόνα 43 “Κλάση LoginActivity”

Η βασικότερη μέθοδος που είναι και αυτή που εκκινείται πρώτη κατά την έναρξη του Activity είναι η **“protected void onCreate(Bundle savedInstanceState)”**, κώδικας της Εικόνας 44, που εκτός των άλλων ορίζει και το Γραφικό Περιβάλλον που θα χρησιμοποιηθεί, γραμμή τρία. Στις γραμμές τέσσερα έως έντεκα συνδέουμε με τη χρήση της μεθόδου **“findviewById()”** τις μεταβλητές που ορίσαμε νωρίτερα με αυτές του Γραφικού Περιβάλλοντος, αρχείο xml, με τη χρήση της κλάσης R, η κλάση στη οποία βρίσκονται τα ID’s όλων των αντικειμένων που βρίσκονται κάτω από τον κατάλογο “/res/”. Στη γραμμή δώδεκα ορίζουμε ότι το αντικείμενο τύπου AutoCompleteTextView **“PortView”**, που χρησιμοποιείται για την δήλωση της πόρτας εξυπηρετητή, θα είναι αριθμός και κατά συνέπεια όταν θα γίνει Focus από τον χρήστη θα ανοίξει ένα αριθμητικό πληκτρολόγιο και όχι ένα αλφαριθμητικό. Στις γραμμές δεκατρία έως δεκαέξι ορίζουμε ότι όταν ο χρήστης πατήσει το κουμπί **“FAQ”** θα ανοίξει μία νέα Activity. Αυτή η Activity είναι τύπου προβολής, **“Intent.ACTION_VIEW”**, δηλαδή το αποτέλεσμα της εκτέλεσής της είναι η προβολή των δεδομένων, η δεύτερη παράμετρος είναι το αντικείμενο που θα προβάλλει. Στη συγκεκριμένη περίπτωση ανοίγει στον φυλλομετρητή μία παρουσίαση που αφορά ένα σύντομο οδηγό χρήσης της εφαρμογής. Αντίστοιχα, στη γραμμή δεκαεπτά, όταν ο χρήστης πατήσει το κουμπί σύνδεση τότε θα εκτελεστεί ο κώδικας που έχουμε ορίσει στη μέθοδο **“private void attemptLogin()”**.

```
1. protected void onCreate(Bundle savedInstanceState) {
2.     super.onCreate(savedInstanceState);
3.     setContentView(R.layout.activity_login);
4.     UsernameView = (AutoCompleteTextView)
   findViewById(R.id.LoginUser);
5.     UrlView = (AutoCompleteTextView)
   findViewById(R.id.LoginServerUser);
6.     PortView = (AutoCompleteTextView)
   findViewById(R.id.LoginPort);
7.     mLoginFormView = findViewById(R.id.login_form);
8.     mProgressView = findViewById(R.id.login_progress);
9.     PasswordView = (EditText) findViewById(R.id.LoginPassword);
10.       FAQ=(Button)findViewById(R.id.FAQB);
11.       Button mEmailSignInButton = (Button)
   findViewById(R.id.email_sign_in_button);
12.       PortView.setInputType(InputType.TYPE_CLASS_NUMBER);
13.       FAQ.setOnClickListener(new View.OnClickListener() {
14.           @Override
15.           public void onClick(View v) {
16.                   Intent intent = new
   Intent(Intent.ACTION_VIEW,
   Uri.parse(getString(R.string.FAQ)));
17.                   startActivity(intent);
18.           }
19.       });
20.       mEmailSignInButton.setOnClickListener(new
   View.OnClickListener() {
21.           @Override
22.           public void onClick(View view) {
23.               attemptLogin();
24.           }
25.       });
26.   }
```

Εικόνα 44 "Μέθοδος onCreate"

Η μέθοδος **"private void attemptLogin()"**, κώδικας της Εικόνας 45, συλλέγει και ετοιμάζει τα στοιχεία εκείνα που χρειάζονται για τη σύνδεση με τη Βάση Δεδομένων. Στις γραμμές δύο έως τέσσερα, ελέγχουμε αν το αντικείμενο τύπου **UserLoginTask** έχει κάποιο περιεχόμενο, έχει ήδη χρησιμοποιηθεί από την **LoginActivity** τότε επιστρέφει χωρίς να εκτελέσει τον κώδικα που ακολουθεί. Σε αντίθετη περίπτωση, γραμμές πέντε έως οχτώ, λαμβάνουμε τις τιμές που έδωσε ο χρήστης. Επειδή η μέθοδος **"getText()"** της κλάσης AutoCompleteTextView επιστρέφει έναν πίνακα χαρακτήρων χρησιμοποιούμε τη μέθοδο **"toString()"** για να πάρουμε την τιμή ολόκληρη ως αλφαριθμητικό. Στην τελευταία μεταβλητή, την πόρτα που αναμένει ο Εξυπηρετητής, επειδή θέλουμε ακέραιο αριθμό αφού κάνουμε την ίδια μετατροπή που κάναμε και στα υπόλοιπα πεδία στο τέλος το μετατρέπουμε σε αριθμό με τη χρήση της μεθόδου **"parseInt(STRING)"** της κλάσης Integer, που αποτελεί μία από τις βασικές κλάσεις της Java. Στη γραμμή εννέα εμφανίζουμε το ProgressBar στο προσκήνιο του Γραφικού μας περιβάλλοντος. Στη συνέχεια γραμμές δέκα και έντεκα, δημιουργούμε ένα νέο στιγμιότυπο τύπου **UserLoginTask**, περνώντας τις κατάλληλες παραμέτρους και μετά το εκτελούμε.

```
private void attemptLogin() {
    if (mAuthTask != null) {
        return;
    }
    String username = UsernameView.getText().toString();
    String password = PasswordView.getText().toString();
    String url = UrlView.getText().toString();
    int port = Integer.parseInt(PortView.getText().toString());
    showProgress(true);
    mAuthTask = new UserLoginTask(username, password, url, port);
    mAuthTask.execute((Void) null);
}
```

Εικόνα 45 "Μέθοδος attemptLogin"

Μετά την εκτέλεση της κλάσης **UserLoginTask** καλείται η μέθοδος **"private void returnMessage(int msg, String UserNi, final String PasswordNi, final String ServerNi, final int PortNi)"**, κώδικας της Εικόνας 46, που έχει ως λειτουργία την εμφάνιση μηνύματος προς τον χρήστη για την επιτυχή ή όχι έκβαση της αυθεντικοποίησης. Δέχεται ως είσοδο τον κωδικό μηνύματος που αφορά αν υπήρξε κάποιο πρόβλημα ή όχι με την αυθεντικοποίηση, το όνομα χρήστη, τον κωδικό χρήστη, τη διεύθυνση του Εξυπηρετητή και τέλος την πόρτα που αναμένει ο Εξυπηρετητής. Στις γραμμές δύο και τρία αρχικοποιούμε τις μεταβλητές μήνυμα και τίτλος που θα χρησιμοποιηθούν ανάλογα με τη μεταβλητή **"msg"** για να εμφανίσουν μήνυμα λάθους ή όχι ανάλογα με την περίπτωση. Οι γραμμές τέσσερα έως επτά είναι η αρχικοποίηση και απόδοση τιμών με βάση τα πεδία που δέχτηκε η μέθοδος ως είσοδο, αξίζει να σημειωθεί ότι μετατρέπουμε τις τιμές σε Final καθώς εντός της μεθόδου **"onClick()"** της **"alertDialogBuilder.setNeutralButton()"** μπορούμε να χρησιμοποιήσουμε μόνο Final μεταβλητές. Στη γραμμή οχτώ γίνεται η αρχικοποίηση του αντικειμένου **"AlertDialog.Builder"** που δημιουργεί ένα alert dialog και θα εμφανίζει μήνυμα επιτυχίας ή όχι. Στη συνέχεια ελέγχουμε τη μεταβλητή **"msg"**, σε περίπτωση επιτυχίας και εφόσον ο χρήστης πατήσει στο μοναδικό κουμπί που διαθέτει το alert dialog, θα αποθηκεύσει τα στοιχεία που έδωσε ο χρήστης και μεταβαίνουμε στην επόμενη Activity. Στη γραμμή δεκατρία δημιουργούμε το αρχείο **DB4OCREDS** με δικαιώματα επεξεργασίας και προβολής αυτού μόνο από την ίδια την εφαρμογή που το δημιούργησε. Για να μπορέσουμε να γράψουμε σε αυτό θα πρέπει να δημιουργήσουμε ένα αντικείμενο τύπου "**SharedPreferences.Editor"** και με τη χρήση της μεθόδου **"putX()"**, όπου **X** ο τύπος της μεταβλητής που θέλουμε να αποθηκεύσουμε. Η αποθήκευση γίνεται σε ζεύγη ***"όνομα_μεταβλητής, τιμή_μεταβλητής"*** και κάνοντας **"commit()"** τις αποθηκεύουμε. Αφού τελειώσουμε και με την αποθήκευση των μεταβλητών τότε ανοίγουμε την επόμενη Activity που είναι η **Initial**. Στην περίπτωση που ο χρήστης δεν έχει δώσει σωστά στοιχεία, κώδικας της Εικόνας 47, εμφανίζεται το αντίστοιχο μήνυμα και όταν

πατήσει το κουμπί δεν θα γίνει απολύτως τίποτα εγκλωβίζοντας τον χρήστη σε αυτή την Activity μέχρι να δώσει τα σωστά στοιχεία σύνδεσης. Αφού τελειώσουμε και με τη λειτουργικότητα του κουμπιού θέτουμε το μήνυμα και τον τίτλο του alert dialog και το εμφανίζουμε.

```
1. private void returnMessage(int msg, String UserNi, final
   String PasswordNi, final String ServerNi, final int PortNi) {
2.     String message = null;
3.     String title = null;
4.     final String uis = UserNi;
5.     final String pass = PasswordNi;
6.     final String serv = ServerNi;
7.     final int po = PortNi;
8.     AlertDialog.Builder alertDialogBuilder = new
   AlertDialog.Builder(this);
9.     if (msg == 0) {
10.          message = getString(R.string.LoginMSGOK);
11.          title = getString(R.string.LoginTitleOK);
12.
   alertDialogBuilder.setNeutralButton(getString(R.string.OkButto
   n), new DialogInterface.OnClickListener() {
13.              @Override
14.              public void onClick(DialogInterface dialog, int
   which) {
15.                  SharedPreferences sharedpreferences =
   getSharedPreferences(getString(R.string.MyPREFERENCES),
   Context.MODE_PRIVATE);
16.                  SharedPreferences.Editor editor =
   sharedpreferences.edit();
17.                  editor.putString(getString(R.string.UserN),
   uis);
18.
   editor.putString(getString(R.string.PasswordN), pass);
19.                  editor.putInt(getString(R.string.PortN), po);
20.                  editor.putString(getString(R.string.ServerN),
   serv);
21.                  editor.commit();
22.                  Intent intent = new
   Intent(LoginActivity.this, Initial.class);
23.                  startActivity(intent);
24.              }
25.          });
```

Εικόνα 46 "Μέθοδος returnMessage (Α΄ Μέρος)"

```
1. } else {
2.         message = getString(R.string.LoginMSGNAK);
3.         title = getString(R.string.LoginTitleNAK);
4.
   alertDialogBuilder.setNeutralButton(getString(R.string.TryAga
   inButton), new DialogInterface.OnClickListener() {
5.             @Override
6.             public void onClick(DialogInterface dialog, int
   which) {
7.                 //Do Nothing
8.             }
9.         });
10.      }
11.      alertDialogBuilder.setTitle(title);
12.      alertDialogBuilder.setMessage(message);
13.      alertDialogBuilder.show();
14.  }
```

Εικόνα 47 "Μέθοδος returnMessage (Β΄ Μέρος)"

Η κλάση **"UserLoginTask"**, κώδικας της Εικόνας 48, κληρονομεί την κλάση **"AsyncTask"**, που αναλαμβάνει να εκτελέσει ένα Thread μέσα από Γραφικό Περιβάλλον και στη συνέχεια επιστρέφει σε αυτό το αποτέλεσμα. Κυρίως χρησιμοποιείται για διαδικασίες που περιέχουν σύνδεση με το δίκτυο και έχει τέσσερα στάδια πριν την εκτέλεση, κατά την εκτέλεση, αλλαγή κατάστασης-πρόοδος και μετά την εκτέλεση. Λόγω της απλότητας της διαδικασίας στην συγκεκριμένη κλάση θα χρησιμοποιήσουμε μόνο τις μεθόδους κατά την εκτέλεση και μετά την εκτέλεση. Αρχικά μέσω του δομήτη αποθηκεύουμε στις τοπικές μεταβλητές το όνομα χρήστη, κωδικό χρήστη, τη διεύθυνση του Εξυπηρετητή και την πόρτα στην οποία αναμένει τα ερωτήματα. Με την εντολή **"mAuthTask.execute()"** καλούμε την **"onPreExecute()"**, την οποία παραλείψαμε καθώς δεν χρειαζόταν στη συγκεκριμένη περίπτωση. Στη συνέχεια καλείται η **"doInBackground()"**, όπου εκτελείται και το Thread.

```
1. public class UserLoginTask extends AsyncTask<Void, Void,
   Boolean> {
2.
3.     private String mUser;
4.     private String mPassword;
5.     private String mServer;
6.     private int mPort;
7.     private int falseAlarm = 0;
8.
9.     UserLoginTask(String user, String password, String server,
   int port) {
10.          mUser = user;
11.          mPassword = password;
12.          mServer = server;
13.          mPort = port;
14.      }
```

Εικόνα 48 "Κλάση UserLoginTask"

Στην μέθοδο **"doInBackground()"**, κώδικας της Εικόνας 49, στη γραμμή δύο δημιουργούμε μία σύνδεση με τη Βάση Δεδομένων χρησιμοποιώντας τα στοιχεία σύνδεσης που μας έδωσε ο χρήστης και στη γραμμή τρία κλείνουμε τη σύνδεση. Έτσι σε περίπτωση που η σύνδεση είναι επιτυχημένη η σύνδεση θα κλείσει και η μεταβλητή **falseAlarm** θα πάρει την τιμή **"0"**, ενώ στην περίπτωση που υπάρξει κάποιο πρόβλημα με τη σύνδεση δεν θα εκτελεστεί η τέταρτη γραμμή, η **"catch()"** θα πιάσει το σφάλμα η μεταβλητή **falseAlarm** θα πάρει την τιμή **"1"** και θα τερματίσει.

```
1. protected Boolean doInBackground(Void... params) {
2.     try {
3.         ObjectContainer db =
   Db4oClientServer.openClient(Db4oClientServer.newClientConfigu
   ration(), mServer, mPort, mUser, mPassword);
4.         db.close();
5.         falseAlarm = 0;
6.     } catch (Exception ex) {
7.         falseAlarm = 1;
8.     }
9.     return true;
10.  }
```

Εικόνα 49 "Μέθοδος doInBackground"

Τέλος, θα εκτελεστεί η μέθοδος **"onPostExecute()"**, κώδικας της Εικόνας 50, που είναι μία από τις μεθόδους που αλληλεπιδρούν με το Γραφικό Περιβάλλον, η άλλη είναι η "onProgressUpdate()" που ενημερώνει το ProgressBar και λόγω της απλότητας του κώδικα δεν χρειάστηκε στη συγκεκριμένη κλάση. Στην μετα την εκτέλεση μέθοδο απελευθερώνουμε την μεταβλητή **"mAuthTask"**, απαλείφουμε από το προσκήνιο το ProgressBar και καλούμε τη μέθοδο **"returnMessage()"** που περιγράψαμε νωρίτερα.

```
1. protected void onPostExecute(final Boolean success) {
2.     mAuthTask = null;
3.     showProgress(false);
4.     returnMessage(falseAlarm, mUser, mPassword, mServer,
   mPort);
5. }
```

Εικόνα 50 "Μέθοδος onPostExecute"

3.2.3 Initial.java

Η κλάση **"Initial"** έπεται της **"LoginActivity"** και έχει ως λειτουργία την παρουσίαση των κλάσεων που έχει αποθηκεύσει ο χρήστης στη Βάση Δεδομένων db4o καθώς και τα πεδία αυτών. Για την καλύτερη προβολή των διαθέσιμων κλάσεων και των πεδίων αυτών αποφασίσαμε να κάνουμε χρήση του NavigationView, κώδικας της Εικόνας 51, που υλοποιεί τη διεπαφή **"NavigationView.OnNavigationItemSelectedListener"** για τη λειτουργία του Drawer και κληρονομεί την κλάση **"AppCompatActivity"**. Στις γραμμές τρία έως

επτά δηλώνουμε τις global μεταβλητές που θα χρησιμοποιήσουμε, την μεταβλητή τύπου **ListView** θα τη χρειαστούμε για να εισάγουμε στο Γραφικό Περιβάλλον μία λίστα από τα πεδία κάθε κλάσης συνοδευόμενα από τον τύπο τους, τη μεταβλητή τύπου **Menu** για την διαχείριση του μενού και των γεγονότων από τη διάδραση του χρήστη με τα αντικείμενα αυτού. Η μεταβλητή **"kClass"** τύπου αλφαριθμητικό χρησιμοποιείται για την αποστολή του ονόματος της κλάσης που έχει επιλέξει ο χρήστης στην Activity **ConstraintsActivity** για την εισαγωγή των περιορισμών, η μεταβλητή **"knownClasses"** τύπου λίστας αλφαριθμητικό χρησιμοποιείται για την προσωρινή αποθήκευση των ονομάτων των κλάσεων με σκοπό την εμφάνισή τους στο Drawer. Τέλος, η μεταβλητή τύπου **Context** χρησιμοποιείται για μπορούμε να διαχειριστούμε το αντικείμενο SharedPreferences από την κλάση **Db4oSubClass**, καθώς αυτή δεν είναι δυνατή δίχως τη χρήση των μεταβλητών/δικαιωμάτων της εκάστοτε Activity.

```
1. public class Initial extends AppCompatActivity
2.         implements
   NavigationView.OnNavigationItemSelectedListener {
3.     private ListView ATTListView;
4.     private Menu menu;
5.     private String kClass = null;
6.     private List<String> knownClasses;
7.     private Context ctx;
```

Εικόνα 51 "Κλάση Initial"

Η μέθοδος **"protected void onCreate(Bundle savedInstanceState)"**, κώδικας της Εικόνας 52, που εκτός των άλλων ορίζει και το Γραφικό Περιβάλλον που θα χρησιμοποιηθεί, στις γραμμές τέσσερα και πέντε αφού συνδέσουμε την επιθυμητή γραμμή εργαλείων με την αντίστοιχη μεταβλητή του προγράμματός μας την ορίζουμε ως την ενεργή γραμμή εργαλείων της Activity μας. Στη γραμμή έξι αποθηκεύουμε ένα instance της Activity στην μεταβλητή τύπου **Context** για χρήση αυτής στην κλάση **Db4oSubClass**. Στις γραμμές επτά έως δεκατρία συνδέουμε το **Floating Button**, είναι το κυκλικό κουμπί το οποίο βρίσκεται πάνω από το Γραφικό Περιβάλλον, λειτουργικότητα του Material Design όπως αναφέραμε στην παράγραφο 1.1.2. Ορίζουμε ότι δεν μπορεί να χρησιμοποιηθεί από τον χρήστη μέχρι να επιλέξει κάποια κλάση, **"setClickable(false)"**, και τέλος θέτουμε τι θα συμβεί στην περίπτωση που ο χρήστης θα πατήσει το συγκεκριμένο κουμπί, η εκκίνηση της Activity **ConstraintsActivity** για την δημιουργία περιορισμών. Πιο συγκεκριμένα στη γραμμή έντεκα περνάμε στην επόμενη Activity την παράμετρο **"kClass"** που είναι το όνομα της κλάσης που επέλεξε να θέσει ως κλάση ρίζα για τους περιορισμούς ο χρήστης και στην γραμμή δώδεκα γίνεται η εκκίνηση της νέας

Activity. Στη γραμμή δεκατέσσερα συνδέουμε το Drawer από το Γραφικό Περιβάλλον με την αντίστοιχη μεταβλητή που έχουμε έτσι ώστε να μπορούμε να το διαχειριστούμε προγραμματιστικά. Στις γραμμές δεκαπέντε έως δεκαοχτώ ορίζουμε το κείμενο που θα εμφανίζεται κατά το άνοιγμα και κλείσιμο του Drawer και συνδέουμε τον κατάλληλο listener. Στις γραμμές που ακολουθούν βλέπουμε με τη σειρά ότι συνδέουμε τη λίστα που θα εμφανίζονται τα πεδία της κλάσης με την μεταβλητή **"ATTListView"**, συνδέουμε το NavigationView του γραφικού περιβάλλοντος με τη μεταβλητή **"navigationView"** για τη ρύθμιση παραμέτρων μέσα από την κλάση Initial, την αρχικοποίηση της λίστας **"knownClasses"** όπου θα αποθηκευτούν προσωρινά τα ονόματα των κλάσεων που υπάρχουν στη Βάση Δεδομένων. Στη γραμμή είκοσι τρία καλούμε και εκτελούμε την κλάση **LoadClassATTTask**, που κληρονομεί την κλάση **AsyncTask** για την εκτέλεση ενός Thread. Στο τέλος της μεθόδου αυτής ορίζουμε τον Listener για το αντικείμενο **"navigationView"** όπου θα θέσουμε την επιθυμητή λειτουργικότητα κάθε φορά που ο χρήστης επιλέγει μία κλάση.

```
1. protected void onCreate(Bundle savedInstanceState) {
2.     super.onCreate(savedInstanceState);
3.     setContentView(R.layout.activity_initial);
4.     Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
5.     setSupportActionBar(toolbar);
6.     ctx = this;
7.     FloatingActionButton fab = (FloatingActionButton)
   findViewById(R.id.proceedToQuery);
8.     fab.setClickable(false);
9.     fab.setOnClickListener(new View.OnClickListener() {
10.          @Override
11.          public void onClick(View view) {
12.              Intent x = new Intent(Initial.this,
   ConstraintsActivity.class);
13.              x.putExtra("className", kClass);
14.              startActivity(x);
15.          }
16.      });
17.      DrawerLayout drawer = (DrawerLayout)
   findViewById(R.id.drawer_layout);
18.      ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
19.              this, drawer, toolbar,
   R.string.navigation_drawer_open,
   R.string.navigation_drawer_close);
20.      drawer.addDrawerListener(toggle);
21.      toggle.syncState();
22.      ATTListView = (ListView)
   findViewById(R.id.fieldsKclasses);
23.      NavigationView navigationView = (NavigationView)
   findViewById(R.id.nav_view);
24.      menu = navigationView.getMenu();
25.      knownClasses = new ArrayList<>();
26.      new LoadClassATTTask().execute();
27.      navigationView.setNavigationItemSelectedListener(this);
28.  }
```

Εικόνα 52 "Μέθοδος onCreate"

Η μέθοδος **“public void onBackPressed()”**, κώδικας της Εικόνας 53, εκτελείται κάθε φορά που ο χρήστης πατάει το κουμπί με τη λειτουργικότητα “πίσω”. Στην περίπτωση που ο χρήστης πατήσει το κουμπί πίσω ενώ ο Drawer είναι στο προσκήνιο τότε τον επαναφέρει στο παρασκήνιο, σε διαφορετική περίπτωση εκτελείται η προκαθορισμένη από το Λειτουργικό Σύστημα ενέργεια που είναι η επιστροφή στην προηγούμενη Activity, σε περίπτωση που το πατήσουμε στην Main / Launcher Activity τότε επιστρέφουμε στην αρχική Activity που χρησιμοποιείται από το Λειτουργικό Σύστημα.

```
1. public void onBackPressed() {
2.     DrawerLayout drawer = (DrawerLayout)
   findViewById(R.id.drawer_layout);
3.     if (drawer.isDrawerOpen(GravityCompat.START)) {
4.         drawer.closeDrawer(GravityCompat.START);
5.     } else {
6.         super.onBackPressed();
7.     }
8. }
```

Εικόνα 53 “Μέθοδος onBackPressed”

Οι μέθοδοι **“public boolean onCreateOptionsMenu(Menu menu)”** και **“public boolean onOptionsItemSelected(MenuItem item)”**, κώδικας της Εικόνας 54, έχουν την ευθύνη να ορίσουν το μενού που θέλουμε να χρησιμοποιήσουμε στη συγκεκριμένη Activity καθώς και το τι θα γίνει σε περίπτωση επιλογής κάποιου αντικειμένου από αυτό. Κάθε φορά που ο χρήστης επιλέγει ένα αντικείμενο από το μενού εκτελείται η μέθοδος **“public Boolean onOptionsItemSelected(MenuItem item)”**, με τη χρήση του **“item.getItemId()”** πέρνουμε το ID του αντικειμένου που επιλέξαμε και στη συνέχεια το αντιπαραβάλουμε με τα ID που έχουμε αποθηκεύσει στο αρχείο R. Στη συγκεκριμένη περίπτωση όταν ο χρήστης ανοίξει το μενού εμφανίζονται δύο επιλογές η αποσύνδεση και η εμφάνιση ενός σύντομου οδηγού χρήσης. Στην πρώτη περίπτωση ανοίγουμε την Activity **LoginActivity**, ωστόσο αξίζει να σημειωθεί ότι στη γραμμή δεκατρία ορίζουμε σημαία ορίζοντας πως θα διαχειριστεί το Λειτουργικό Σύστημα το συγκεκριμένο συμβάν. Το **“Intent.FLAG_ACTIVITY_CLEAR_TOP”** σημαίνει ότι όλες οι Activity που ενεργοποιήθηκαν μετά από την Activity που θέλουμε να πάμε και πριν την Activity που βρισκόμαστε, συμπεριλαμβανομένης και αυτής που βρισκόμαστε, θα τερματιστούν πριν την έναρξη της Activity που θέλουμε να πάμε. Στη δεύτερη περίπτωση θα ανοίξει ένας φυλλομετρητής με την παρουσίαση ενός σύντομου οδηγού χρήσης της εφαρμογής.

```
1. public boolean onCreateOptionsMenu(Menu menu) {
2.     getMenuInflater().inflate(R.menu.initial, menu);
3.     return true;
4. }
5. public boolean onOptionsItemSelected(MenuItem item) {
6.     int id = item.getItemId();
7.     if (id == R.id.logout) {
8.         Intent intent = new Intent( this, LoginActivity.class
   );
9.         intent.setFlags( Intent.FLAG_ACTIVITY_CLEAR_TOP );
10.           startActivity(intent);
11.           return true;
12.       }
13.       if (id == R.id.FAQ) {
14.           Intent intent = new Intent(Intent.ACTION_VIEW,
   Uri.parse(getString(R.string.FAQ)));
15.           startActivity(intent);
16.           return true;
17.       }
18.       return super.onOptionsItemSelected(item);
19.   }
```

Εικόνα 54 "Μέθοδοι onCreateOptionsMenu και onOptionsItemSelected"

Η μέθοδος **"private void Class2Text()"**, κώδικας της Εικόνας 55, είναι μία βοηθητική μέθοδος που σκοπός της είναι το γέμισμα του Drawer με τα ονόματα των κλάσεων. Σε αντίθεση με το Navigation Drawer που υλοποιείται με τη χρήση μίας λίστας, στο Navigation View θα πρέπει να περάσουμε το κάθε αντικείμενο ξεχωριστά. Για κάθε αλφαριθμητικό αντικείμενο που βρίσκεται στη λίστα **"knownClasses"** το τοποθετούμε στο μενού με τη χρήση της μεθόδου **"add()"** του αντικειμένου **Menu**. Στη συνέχεια θέτουμε ως επιλεγμένο το πρώτο στοιχείο που βρίσκεται στη λίστα αυτή.

```
1. private void Class2Text() {
2.     for (String s : knownClasses) {
3.         menu.add(s);
4.     }
5.     onNavigationItemSelected(menu.getItem(0));
6. }
```

Εικόνα 55 "Μέθοδος Class2Text"

Η μέθοδος **“public boolean onNavigationItemSelected(MenuItem item)”**, κώδικας της Εικόνας 56, εκτελείται κάθε φορά που επιλέγεται μία κλάση. Κατά την επιλογή της κλάσης ενημερώνουμε την τιμή τόσο της μεταβλητής **“kClass”** όσο και του τίτλου που εμφανίζεται στο **ActionBar** της Activity. Δεδομένου ότι ο χρήστης έχει επιλέξει κάποια κλάση του επιτρέπουμε να προβεί στον ορισμό περιορισμών, θέτοντας ως **clickable** το **Floating Button**, γραμμή πέντε. Στη συνέχεια, γραμμή έξι, καλούμε και εκτελούμε την κλάση **LoadObjectsATTTask** που θα εκτελέσει το Thread για τη λήψη των πεδίων της επιλεγμένης κλάσης. Τέλος, κλείνουμε τον Drawer για να μπορέσει ο χρήστης να αλληλεπιδράσει με το περιεχόμενο της κύριας Activity, τα πεδία της κλάσης.

```
1. public boolean onNavigationItemSelected(MenuItem item) {
2.     kClass = item.getTitle().toString();
3.     setTitle(kClass);
4.     FloatingActionButton proceedToQuery =
   (FloatingActionButton) findViewById(R.id.proceedToQuery);
5.     proceedToQuery.setClickable(true);
6.     new LoadObjectsATTTask().execute();
7.     DrawerLayout drawer = (DrawerLayout)
   findViewById(R.id.drawer_layout);
8.     drawer.closeDrawer(GravityCompat.START);
9.     return true;
10.    }
```

Εικόνα 56 “Μέθοδος onNavigationItemSelected”

Στην εσωτερική κλάση **“LoadObjectsATTTask”**, κώδικας της Εικόνας 57, θα κάνουμε χρήση των μεθόδων πριν την εκτέλεση, κατά την εκτέλεση και μετά την εκτέλεση. Στην μέθοδο **“onPreExecute()”** αρχικοποιούμε τις μεταβλητές που θα χρησιμοποιήσουμε, τη λίστα τύπου αλφαριθμητικό **“reflectATTListSTRING”**, και στη συνέχεια αφού αρχικοποιήσουμε το αντικείμενο τύπου ProgressDialog για χρήση στην συγκεκριμένη Activity θέτουμε το μήνυμα που θα εμφανίζεται και τέλος το παρουσιάζουμε στο προσκήνιο, αυτό το κάνουμε καθώς ανάλογα με την ποιότητα της σύνδεσης αλλά και με τον αριθμό των αντικειμένων που βρίσκονται στη Βάση Δεδομένων η διαδικασία λήψης μπορεί να διαρκέσει λίγα δευτερόλεπτα και οφείλουμε να ενημερώσουμε σχετικά με την πορεία της διαδικασίας τον χρήστη. Στην μέθοδο **“doInBackground()”** εκτελείται η σύνδεση με τη βάση, η λήψη των αποτελεσμάτων και ο τερματισμός της σύνδεσης. Αυτό συμβαίνει με τη χρήση της κλάσης Db4oSubClass, αρχικά δημιουργούμε ένα αντικείμενο τύπου **Db4oSubClass** περνώντας παραμετρικά τη μεταβλητή τύπου **Context** για τη διαχείριση των SharedPreferences από την Db4oSubClass. Μετά με τη χρήση της μεθόδου **“reflectFieldsNameANDTypeListSTRING()”** και περνώντας

παραμετρικά το όνομα της επιλεγμένης κλάσης ως αλφαριθμητικό λαμβάνουμε τα ονόματα των πεδίων της κλάσης αυτής συνοδευόμενα με τον τύπο δεδομένων τους σε μορφή λίστας αλφαριθμητικών. Με τη βοήθεια της **"CloseDB()"** τερματίζουμε τη σύνδεση με τον Εξυπηρετητή. Στη συνέχεια η μέθοδος **"onPostExecute()"** αναλαμβάνει να προβάλλει στο Γραφικό Περιβάλλον της Activity τα δεδομένα που έλαβε από την **"doInBackground()"**, αυτό γίνεται με τη χρήση ενός Adapter, γραμμή είκοσι. Στο τέλος της μεθόδου αυτής τερματίζουμε και επαναφέρουμε στο παρασκήνιο το αντικείμενο τύπου ProgressDialog.

```
1.   private class LoadObjectsATTTask extends AsyncTask<Integer,
   String, String> {
2.         List<String> reflectATTListSTRING;
3.         ProgressDialog mProgressDialog;
4.         protected String doInBackground(Integer... params) {
5.             Db4oSubClass db4oSubClass = new Db4oSubClass(ctx);
6.             reflectATTListSTRING =
   db4oSubClass.reflectFieldsNameANDTypeListSTRING(kClass);
7.             db4oSubClass.CloseDB();
8.             return null;
9.         }
10.          protected void onPreExecute() {
11.              super.onPreExecute();
12.              reflectATTListSTRING = new ArrayList<>();
13.              mProgressDialog = new
   ProgressDialog(Initial.this);
14.              mProgressDialog.setIndeterminate(true);
15.              mProgressDialog.setTitle("Loading Fields ");
16.              mProgressDialog.setMessage("Searching for the
   required Fields");
17.              mProgressDialog.show();
18.          }
19.          protected void onPostExecute(String t) {
20.              ATTListView.setAdapter(new
   ReflectFieldsListViewAdapter(Initial.this,
   reflectATTListSTRING, knownClasses));
21.              mProgressDialog.dismiss();
22.          }
23.      }
```

Εικόνα 57 "Κλάση LoadObjectsATTTask"

Στη συνέχεια έχουμε την εσωτερική κλάση **LoadClassATTTask**, κώδικας της εικόνας 58, όπου θα κάνουμε χρήση των μεθόδων πριν την εκτέλεση, κατά την εκτέλεση και μετά την εκτέλεση. Στην μέθοδο **"onPreExecute()"** αρχικοποιούμε τη μεταβλητή που θα χρησιμοποιήσουμε, το αντικείμενο τύπου ProgressDialog για χρήση στην συγκεκριμένη Activity θέτουμε το μήνυμα που θα εμφανίζεται και τέλος το παρουσιάζουμε στο προσκήνιο. Στην μέθοδο **"doInBackground()"** εκτελείται η σύνδεση με τη βάση, η λήψη των αποτελεσμάτων και ο τερματισμός της σύνδεσης. Με τη χρήση της μεθόδου **"reflectClassesAsSTR()"** λαμβάνουμε τα ονόματα των κλάσεων που βρίσκονται αποθηκευμένα στη Βάση Δεδομένων και έχουν δημιουργηθεί από τον χρήστη, όπως αναφέραμε στην παράγραφο 3.1.1, σε μορφή

λίστας αλφαριθμητικό. Με τη βοήθεια της **“CloseDB()”** τερματίζουμε τη σύνδεση με τον Εξυπηρετητή. Στην μέθοδο **“onPostExecute()”** καλούμε τη μέθοδο **“Class2Text()”**, για την εμφάνιση των κλάσεων όπως είδαμε νωρίτερα και τερματίζουμε και επαναφέρουμε στο παρασκήνιο το αντικείμενο τύπου ProgressDialog.

```
1. private class LoadClassATTTask extends AsyncTask<Integer,
   String, String> {
2.     ProgressDialog mProgressDialog;
3.     protected String doInBackground(Integer... params) {
4.         Db4oSubClass db4oSubClass = new Db4oSubClass(ctx);
5.         knownClasses = db4oSubClass.reflectClassesAsSTR();
6.         db4oSubClass.CloseDB();
7.         return null;
8.     }
9.     protected void onPreExecute() {
10.          super.onPreExecute();
11.          mProgressDialog = new ProgressDialog(Initial.this);
12.          mProgressDialog.setIndeterminate(true);
13.          mProgressDialog.setTitle("Loading Clases ");
14.          mProgressDialog.setMessage("Searching for
   Classes");
15.          mProgressDialog.show();
16.      }
17.      protected void onPostExecute(String t) {
18.          Class2Text();
19.          mProgressDialog.dismiss();
20.      }
21.  }
```

Εικόνα 58 “Κλάση LoadClassATTTask”

3.2.4 ConstraintsActivity.java

Η κλάση **“ConstraintsActivity”**, κώδικας της Εικόνας 59, κληρονομεί την κλάση **“AppCompatActivity”**. Στις γραμμές δύο έως εννέα ορίζουμε τις global μεταβλητές που θα χρησιμοποιήσουμε. Με τη σειρά που εμφανίζονται αυτές είναι η μεταβλητή **“reflectFieldsRecyclerView”** τύπου RecyclerView, το RecyclerView αποτελεί μία πιο ανεπτυγμένη και εύχρηστη έκδοση του ListView δίνοντας την δυνατότητα για την προβολή μεγάλου όγκου δεδομένων χρησιμοποιώντας κάθε φορά μόνο τον απαραίτητο αριθμό από αντικείμενα. Αυτό καθιστά το RecyclerView ιδανικό για περιπτώσεις όπου το περιεχόμενο που χρειάζεται να προβληθεί μεταβάλλεται δυναμικά. Στο RecyclerView, σε αντίθεση με το ListView όπου η δημιουργία custom adapter είναι προαιρετική, στο RecyclerView απαιτείται custom adapter όπως η μεταβλητή **“reflectFieldsRecyclerViewAdapter”** τύπου ReflectFieldsRecyclerViewAdapter για την εμφάνιση των δεδομένων ενός Dataset

στα αντικείμενα ενός RecyclerView. Η μεταβλητή **“reflectClassName”** είναι τύπου αλφαριθμητικό και αφορά στον τύπο της μεταβλητής που είναι αναφορά σε άλλο αντικείμενο αποθηκευμένο στη Βάση Δεδομένων που επέλεξε ο χρήστης για να θέσει κάποιο περιορισμό. Η μεταβλητή **“classPath”** τύπου αλφαριθμητικό έχει την διαδρομή (path) έως αυτή την κλάση. Για παράδειγμα, αν ο χρήστης έχει επιλέξει στην κλάση Initial την κλάση Participates και στη συνέχεια το πεδίο athlete και στη συνέχεια το πεδίο trainer,που αποτελεί αναφορά σε αντικείμενο τύπου Trainer, τότε η τιμή της μεταβλητής “classPath” θα είναι Participates.athlete.trainer. Η λίστα αλφαριθμητικό **“userClasses”** θα φιλοξενήσει τα ονόματα όλων των κλάσεων που είναι αποθηκευμένα στην Βάση Δεδομένων και έχουν δημιουργηθεί από τον χρήστη. Καθώς ο χρήστης δύναται να θέσει μία σειρά από περιορισμούς άγνωστους στον αριθμό χρησιμοποιήσαμε μία λίστα τύπου **MyConstraint**, είδαμε την κλάση MyConstraint στην παράγραφο 3.1.4, για να αποθηκεύσουμε όλους τους περιορισμούς του χρήστη. Η μεταβλητή **“mapper”** τύπου ObjectMapper θα χρησιμοποιηθεί για την μετατροπή αντικειμένων Java σε αντικείμενα JSON και αντίστροφα τα οποία περνάμε παραμετρικά από Activity σε Activity [19]. Τέλος η μεταβλητή **“ctx”** τύπου Context, θα χρησιμοποιηθεί για να περάσουμε παραμετρικά στην κλάση **Db4oSubClass** ένα instance της Activity **ConstraintsActivity** για να είναι δυνατή η προσπέλαση των SharedPreferences από την κλάση Db4oSubClass.

```
1. public class ConstraintsActivity extends AppCompatActivity {
2.
3.     private RecyclerView reflectFieldsRecyclerView;
4.     private ReflectFieldsRecyclerViewAdapter
   reflectFieldsRecyclerViewAdapter;
5.     private String reflectClassName;
6.     private String classPath;
7.     private List<String> userClasses;
8.     private List<MyConstraint> myConstraints;
9.     private static ObjectMapper mapper = new ObjectMapper();
10.      private Context ctx;
```

Εικόνα 59 “Κλάση ConstraintsActivity”

Στη μέθοδο **“protected void onCreate(Bundle savedInstanceState)”**, κώδικας της Εικόνας 60, στις γραμμές τέσσερα και πέντε αφού συνδέσουμε την επιθυμητή γραμμή εργαλείων με την αντίστοιχη μεταβλητή του προγράμματός μας, την ορίζουμε ως την ενεργή γραμμή εργαλείων της Activity μας. Στη γραμμή έξι αποθηκεύουμε ένα instance της Activity στην μεταβλητή τύπου **Context** για χρήση αυτής στην κλάση **Db4oSubClass**. Στις γραμμές επτά και οχτώ αρχικοποιούμε τις δύο λίστες, τύπου αλφαριθμητικό και MyConstraint αντίστοιχα. Επειδή το αντικείμενο RecyclerView σε αντιδιαστολή με τα στοιχεία ListView και Gridview επιτρέπει στον χρήστη να χρησιμοποιήσει custom επιλογή για το τρόπο που θα

προβάλλονται τα αντικείμενα παιδιά που φιλοξενεί ένα RecyclerView, θα πρέπει να ορίσουμε αυτό το layout χρησιμοποιώντας τις μεθόδους **“reflectFieldsRecyclerView.setLayoutManager()”**, που δέχεται ώς παράμετρο αντικείμενα τύπου **RecyclerView.LayoutManager**, και **“reflectFieldsRecyclerView.addItemDecoration()”**, που δέχεται ως παράμετρο αντικείμενα τύπου **RecyclerView.ItemDecoration** όπως είναι η κλάση **DividerItemDecoration** που είδαμε στην παράγραφο 3.1.6, όπως φαίνεται στις γραμμές εννέα έως έντεκα. Στις γραμμές δώδεκα έως δεκαπέντε ανακτούμε την τιμή του πεδίου **“className”** που έχουμε περάσει παραμετρικά από την Activity που εκκίνησε αυτή την Activity και είναι τύπου αλφαριθμητικό για αυτό το λόγο χρησιμοποιούμε τη μέθοδο **“getIntent().getStringExtra()”**. Στη συνέχεια αφού βεβαιωθούμε ότι η μεταβλητή **“reflectClassName”** περιέχει κάποια έγκυρη τιμή καλούμε και εκτελούμε την κλάση **GetReflectFields** περνώντας παραμετρικά το όνομα της κλάσης της οποίας τα πεδία θέλουμε να μας επιστραφούν. Στις γραμμές δεκαέξι έως είκοσι ένα, ανακτούμε την τιμή του πεδίου **“classPath”** που έχουμε περάσει παραμετρικά από την Activity που εκκίνησε αυτή την Activity και είναι τύπου αλφαριθμητικό. Στη συνέχεια ελέγχουμε ότι η μεταβλητή **“classPath”** περιέχει κάποια κάποια τιμή και θέτει ως τίτλο στο **ActionBar** της Activity την τιμή του πεδίου **“classPath”** και μέσα σε παρενθέσεις τον τύπο της μεταβλητής που επιλέξαμε, σε αντίθετη περίπτωση εμφανίζεται μόνο η τιμή της μεταβλητής **“reflectClassName”**. Στις γραμμές είκοσι δύο μέχρι τριάντα ανακτούμε την τιμή του πεδίου **“ConstraintsJsonData”** που έχουμε περάσει παραμετρικά από την Activity που εκκίνησε αυτή την Activity και είναι τύπου. Στη συνέχεια ελέγχουμε ότι η μεταβλητή **“jsonData”** περιέχει κάποια τιμή και με τη χρήση του αντικειμένου ObjectMapper μεταφράζουμε το JSON αρχείο σε αντικείμενο τύπου **ConstraintsJsonData**.

```
1. protected void onCreate(final Bundle savedInstanceState) {
2.     super.onCreate(savedInstanceState);
3.     setContentView(R.layout.activity_constraints);
4.     Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
5.     setSupportActionBar(toolbar);
6.     ctx = this;
7.     userClasses = new ArrayList<>();
8.     myConstraints = new ArrayList<>();
9.     reflectFieldsRecyclerView = (RecyclerView)
   findViewById(R.id.reflectFieldsRecyclerView);
10.      reflectFieldsRecyclerView.setLayoutManager(new
   LinearLayoutManager(this));
11.      reflectFieldsRecyclerView.addItemDecoration(new
   DividerItemDecoration(this));
12.      reflectClassName =
   getIntent().getStringExtra(getString(R.string.CLASS_NAME));
13.      if (reflectClassName != null) {
14.          new GetReflectFields().execute(reflectClassName);
15.      }
16.      classPath =
   getIntent().getStringExtra(getString(R.string.CLASS_PATH));
17.      if (classPath != null) {
18.          setTitle(classPath + " (" + reflectClassName + ")");
19.      } else {
20.          setTitle(reflectClassName);
21.      }
22.      String jsonData =
   getIntent().getStringExtra(getString(R.string.ConsJSOND));
23.      if (jsonData != null) {
24.          try {
25.              ConstraintsJsonData constraintsJsonData =
   mapper.readValue(jsonData, ConstraintsJsonData.class);
26.              myConstraints =
   constraintsJsonData.getConstraints();
27.          } catch (IOException e) {
28.              e.printStackTrace();
29.          }
30.      }
```

Εικόνα 60 "Μέθοδος onCreate (Α' Μέρος)"

Στα κομμάτια κώδικα των Εικόνων 61 έως 63, ορίζουμε την λειτουργικότητα του στοιχείου **FloatingActionButton** και πιο συγκεκριμένα όταν το πατήσει ο χρήστης. Και στις δύο περιπτώσεις όταν ο χρήστης θα πατήσει αυτό το κουμπί θα μεταφερθεί στην Activity **RecursivePrint**. Στην πρώτη περίπτωση αν ο χρήστης έχει ορίσει πέραν των δύο περιορισμούς τότε εμφανίζεται ένα απλό alert dialog στο οποίο ο χρήστης καλείται να επιλέξει αν θα εκτελέσει όλους τους περιορισμούς με τη χρήση λογικού "και" ή με τη χρήση λογικού "ή". Στη συνέχεια η επιλογή του χρήστη αποθηκεύεται στο πεδίο **"operator"** της κλάσης ConstraintsJsonData μαζί με τη λίστα των περιορισμών **"myConstraints"**. Συνεχίζοντας να χτίζουμε το Intent για την κλάση RecursivePrint με τη χρήση της μεθόδου **"intent.putExtra()"**, ορίζουμε τα παρακάτω πεδία. Το πεδίο **"QueryFlag"** ενημερώνει την κλάση RecursivePrint αν θα εμφανίσει ένα αντικείμενο ή μια λίστα από αντικείμενα του ίδιου τύπου. Το πεδίο **"className"** ανταποκρίνεται στη διαδρομή (path) του πρώτου περιορισμού. Στην επόμενη γραμμή με τη χρήση του αντικειμένου **ObjectMapper** μετασχηματίζουμε το αντικείμενο τύπου **ConstraintsJsonData** σε

τύπου JSON για να το περάσουμε παραμετρικά στην Activity **RecursivePrint**. Το πεδίο **"reflectClassIndex"** αναφέρεται στην θέση που έχει το αντικείμενο που επιλέξαμε στην λίστα των αποτελεσμάτων από την εκτέλεση του ερωτήματος (query), αυτό το πεδίο χρησιμοποιείται μόνο στην κλάση RecursivePrint και για αυτό το λόγο πέρνει την τιμή **"-1"** εδώ. Στην περίπτωση που ο χρήστης δεν έχει επιλέξει κάποιο περιορισμό τότε περνάμε παραμετρικά μόνο τα πεδία **"QueryFlag"**, **"className"** και **"reflectClassIndex"**. Στο τέλος της μεθόδου **"onCreate()"** ορίζουμε το αντικείμενο **ActionBar** που θα χρησιμοποιηθεί απο την Activity.

```
1. FloatingActionButton fab = (FloatingActionButton)
   findViewById(R.id.fab);
2. if (fab != null) {
3.    fab.setOnClickListener(new View.OnClickListener() {
4.        @Override
5.        public void onClick(View view) {
6.            final Intent intent = new
   Intent(ConstraintsActivity.this, RecursivePrint.class);
7.            if (myConstraints.size() == 1) {
8.                ConstraintsJsonData constraintsJsonData = new
   ConstraintsJsonData();
9.
   constraintsJsonData.setConstraints(myConstraints);
10.                constraintsJsonData.setOperator(0);
11.                try {
12.                    intent.putExtra("QueryFlag", true);
13.
   intent.putExtra(getString(R.string.CLASS_NAME),
   constraintsJsonData.getConstraints().get(0).getPath().get(0));
14.    intent.putExtra(getString(R.string.ConsJSOND),
   mapper.writeValueAsString(constraintsJsonData));
15.                    intent.putExtra("reflectClassIndex", -1);
16.                } catch (JsonProcessingException e) {
17.                    e.printStackTrace();
18.                }
19.                startActivity(intent);
20.            }
```

Εικόνα 61 "Μέθοδος onCreate (Β' Μέρος)"

```
1. else if (!myConstraints.isEmpty()) {
2.     final String[] operators = new String[]{"AND", "OR"};
3.     AlertDialog.Builder builder = new
   AlertDialog.Builder(ConstraintsActivity.this);
4.     builder.setTitle("Select operator")
5.             .setItems(operators, new
   DialogInterface.OnClickListener() {
6.                 public void onClick(DialogInterface dialog, int
   which) {
7.                     ConstraintsJsonData constraintsJsonData =
   new ConstraintsJsonData();
8.
   constraintsJsonData.setConstraints(myConstraints);
9.                     constraintsJsonData.setOperator(which);
10.                        try {
11.                            intent.putExtra("QueryFlag", true);
12.
   intent.putExtra(getString(R.string.CLASS_NAME),
   constraintsJsonData.getConstraints().get(0).getPath().get(0));
13.
   intent.putExtra(getString(R.string.ConsJSOND),
   mapper.writeValueAsString(constraintsJsonData));
14.                            intent.putExtra("reflectClassIndex",
   -1);
15.                        } catch (JsonProcessingException e) {
16.                            e.printStackTrace();
17.                        }
18.                        startActivity(intent);
19.                    }
20.                });
21.     builder.create().show();
22.   }
```

Εικόνα 62 "Μέθοδος onCreate (Γ' Μέρος)"

```
1.     else {
2.        intent.putExtra("QueryFlag", true);
3.        if (classPath != null) {
4.
   intent.putExtra(getString(R.string.CLASS_NAME),
   classPath.split("\\.")[0]);
5.                } else {
6.
   intent.putExtra(getString(R.string.CLASS_NAME),
   reflectClassName);
7.                }
8.                intent.putExtra("reflectClassIndex", -1);
9.                startActivity(intent);
10.              }
11.          }
12.      });
13.   }
14.   ActionBar actionBar = getSupportActionBar();
15.   if (actionBar != null) {
16.      actionBar.setDisplayHomeAsUpEnabled(true);
17.   }
```

Εικόνα 63 "Μέθοδος onCreate (Α' Μέρος)"

Οι μέθοδοι **"public boolean onCreateOptionsMenu(Menu menu)"** και **"public boolean onOptionsItemSelected(MenuItem item)"**, κώδικας της Εικόνας 64, έχουν την ευθύνη να ορίσουν το μενού που θέλουμε να χρησιμοποιήσουμε στη συγκεκριμένη Activity και το τι θα γίνει σε περίπτωση επιλογής κάποιου αντικειμένου από αυτό. Στη συγκεκριμένη περίπτωση όταν ο χρήστης ανοίξει το μενού εμφανίζονται τέσσερεις επιλογές η επιστροφή στο προηγούμενο View του RecyclerView, η αποσύνδεση, η εμφάνιση ενός σύντομου οδηγού χρήσης και η εμφάνιση των περιορισμών που έχει δηλώσει ο χρήστης. Στην πρώτη περίπτωση όταν ο χρήστης πατήσει το κουμπί θα γυρίσει στο προηγούμενο View, δηλαδή στην κλάση της οποίας επέλεξε το αντικείμενο αναφοράς που έδειχνε στην κλάση που βρίσκεται τώρα. Στην δεύτερη περίπτωση ανοίγουμε την Activity LoginActivity, ωστόσο αξίζει να σημειωθεί ότι στη γραμμή δεκατρία δηλώνουμε μια σημαία ορίζοντας πως θα διαχειριστεί το λειτουργικό σύστημα το συγκεκριμένο συμβάν. Στη τρίτη περίπτωση θα ανοίξει ένας φυλλομετρητής με την παρουσίαση ενός σύντομου οδηγού χρήσης της εφαρμογής. Στην τέταρτη περίπτωση αν δεν έχει οριστεί κάποιος περιορισμός εμφανίζεται σχετικό μήνυμα, σε αντίθετη περίπτωση με τη χρήση του αντικειμένου **ObjectMapper** μετασχηματίζουμε το αντικείμενο τύπου **ConstraintsJsonData** σε τύπου JSON για να το περάσουμε παραμετρικά στην Activity **WatchMyConstraints**.

```
1. public boolean onCreateOptionsMenu(Menu menu) {
2.     getMenuInflater().inflate(R.menu.recursive_print, menu);
3.     return true;
4. }
5. public boolean onOptionsItemSelected(MenuItem item) {
6.     int id = item.getItemId();
7.     if (id == android.R.id.home) {
8.         if (!super.onOptionsItemSelected(item)) {
9.             onBackPressed();
10.         }
11.         return true;
12.     }
13.     if (id == R.id.logout) {
14.         Intent intent = new Intent(this,
    LoginActivity.class);
15.         intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
16.         startActivity(intent);
17.         return true;
18.     }
19.     if (id == R.id.FAQ) {
20.         Intent intent = new Intent(Intent.ACTION_VIEW,
    Uri.parse(getString(R.string.FAQ)));
21.         startActivity(intent);
22.         return true;
23.     }
24.     if (id == R.id.MyConst) {
25.         if (myConstraints.isEmpty()) {
26.             Toast.makeText(ConstraintsActivity.this,"No
    Constraints", Toast.LENGTH_LONG).show();
27.         } else {
28.             try {
29.                 ConstraintsJsonData constraintsJsonData = new
    ConstraintsJsonData();
30.
    constraintsJsonData.setConstraints(myConstraints);
31.                 Intent intent = new
    Intent(ConstraintsActivity.this, WatchMyConstraints.class);
32.
    intent.putExtra(getString(R.string.ConsJSOND),
    mapper.writeValueAsString(constraintsJsonData));
33.                 startActivity(intent);
34.                 return true;
35.             } catch (Exception ex) {
36.             }
37.         }
38.     }
```

Εικόνα 64 "Μέθοδοι onCreateOptionsMenu και onOptionsItemSelected"

Η μέθοδος **"public void onBackPressed()"**, κώδικας της Εικόνας 65, εκτελείται κάθε φορά που ο χρήστης πατάει το κουμπί με τη λειτουργικότητα "πίσω". Στη συγκεκριμένη περίπτωση έχουμε θέσει διαφορετική λειτουργικότητα από την προβλεπόμενη. Εδώ όταν ο χρήστης πατήσει το κουμπί "πίσω" θα μετατρέψουμε με τη χρήση του αντικειμένου τύπου **ObjectMapper** το αντικείμενο **"constraintsJsonData"** σε αρχείο τύπου JSON. Στη συνέχεια θα δημιουργήσουμε ένα **Intent** για να επιστρέψουμε αποτέλεσμα στην Activity που εκκίνησε αυτό το Activity χρησιμοποιώντας και τη σταθερά **"RESULT_OK"** που σημαίνει ότι το Intent εκκινήθηκε, εκτελέστηκε και τερματίστηκε επιτυχώς. Η μέθοδος **"finish()"** τερματίζει την Activity που εκτελείται και την τοποθετεί στο παρασκήνιο.

```
1. public void onBackPressed() {
2.     ConstraintsJsonData constraintsJsonData = new
   ConstraintsJsonData();
3.     constraintsJsonData.setConstraints(myConstraints);
4.     Intent intent = new Intent();
5.     try {
6.         intent.putExtra(getString(R.string.ConsJSOND),
   mapper.writeValueAsString(constraintsJsonData));
7.         Log.i("MyConstraintsActivity",
   mapper.writerWithDefaultPrettyPrinter().writeValueAsString(co
   nstraintsJsonData));
8.     } catch (JsonProcessingException e) {
9.         e.printStackTrace();
10.      }
11.      setResult(RESULT_OK, intent);
12.      finish();
13.  }
```

Εικόνα 65 "Μέθοδος onBackPressed"

Η μέθοδος **"protected void onActivityResult(int requestCode, int resultCode, Intent data)"**, κώδικας της Εικόνας 66, καλείται όταν η Activity που είχαμε εκκινήσει τερματιστεί και πριν το Λειτουργικό Σύστημα καλέσει την μέθοδο **"onResume()"**. Στην περίπτωση που η Activity που επιστρέφεται είναι η Activity που θέλουμε **(requestCode=Constants.REQUEST_CODE)** τότε ελέγχουμε αν έχει ολοκληρωθεί επιτυχώς, δηλαδή **resultCode= -1**, και ανακτούμε την τιμή της παραμέτρου **ConstraintsJsonData** με τη χρήση του αντικειμένου τύπου **ObjectMapper** που περάσαμε παραμετρικά με το πάτημα του κουμπιού πίσω.

```
1. protected void onActivityResult(int requestCode, int
   resultCode, Intent data) {
2.     super.onActivityResult(requestCode, resultCode, data);
3.     if (requestCode == Constants.REQUEST_CODE) {
4.         if (resultCode == AppCompatActivity.RESULT_OK) {
5.             String jsonData =
   data.getExtras().getString(getString(R.string.ConsJSOND));
6.             if (jsonData != null) {
7.                 try {
8.                     ConstraintsJsonData constraintsJsonData =
   mapper.readValue(jsonData, ConstraintsJsonData.class);
9.                     myConstraints =
   constraintsJsonData.getConstraints();
10.                } catch (IOException e) {
11.                    e.printStackTrace();
12.                }
13.            }
14.        }
15.    }
16. }
```

Εικόνα 66 "Μέθοδος onActivityResult"

Η μέθοδος **"private void showReflectFields(List<ReflectField> reflectFields)"**, κώδικας της Εικόνας 67, δέχεται ώς είσοδο μία λίστα τύπου **ReflectFields** και υλοποιεί τη λειτουργικότητα για το τι θα συμβεί σε περίπτωση που ο χρήστης προβεί στην ενέργεια **"onLongClick()"** και την αποθηκεύουμε στον αντίστοιχο **Adapter**. Στην περίπτωση που το αντικείμενο που επιλεγεί είναι τύπου Collection, AbstractCollection, AbstractList, AbstractSequentialList, LinkedList, ArrayList, AbstractSet, HashSet, LinkedHashSet, TreeSet, AbstractMap, HashMap, TreeMap, WeakHashMap, LinkedHashMap, IdentityHashMap, Vector, Stack, Dictionary, Hashtable, Properties, BitSet, ή Πίνακας τότε εμφανίζεται μήνυμα ότι δεν είναι δυνατός ο ορισμός περιορισμού στο πεδίο αυτό. Στην περίπτωση που το επιλεγμένο πεδίο είναι αναφορά σε άλλο αντικείμενο αποθηκευμένο στη Βάση Δεδομένων, **"userclass.contains()"**, τότε εκκινούμε μία νέα Activity **ConstraintsActivity** όπου αυτή τη φορά θα προβάλλουμε τα πεδία της κλάσης στην οποία αναφέρεται το πεδίο που επέλεξε ο χρήστης. Στη συνέχεια με τη χρήση του αντικειμένου τύπου **ObjectMapper** σειριοποιούμε το αντικείμενο του τύπου **ConstraintsJsonData** σε αρχείο μορφής JSON. Εκκινούμε την νέα Activity με τη χρήση της μεθόδου **"startActivityForResult(intent, Constants.REQUEST_CODE)"**, όπου δίνουμε ένα μοναδικό αριθμητικό αναγνωριστικό στο συγκεκριμένο Intent **(Constants.REQUEST_CODE)**. Με τη χρήση της "startActivityForResult" αναμένουμε την επιστροφή κάποιου αποτελέσματος μετά το πέρας του νέου Intent. Σε διαφορετική περίπτωση, κώδικας των Εικόνων 68 και 69, που ο χρήστης επιλέξει πεδίο που δεν

συμπεριλαμβάνεται στις παραπάνω κατηγορίες τότε θα δημιουργηθεί ένα dialog fragment τύπου **ConstraintDialogFragment**, θα δούμε την λειτουργικότητα αυτής της κλάσης σε ακόλουθη παράγραφο, που όταν ο χρήστης πατήσει το κουμπί αποθήκευσης τότε δημιουργείται ένα νέο αντικείμενο τύπου **MyConstraint** και ορίζουμε τις τιμές των πεδίων του ανάλογα με τις επιλογές που έκανε ο χρήστης στην κλάση ConstraintDialogFragment, το οποίο στη συνέχεια το προσθέτουμε στη λίστα των περιορισμών. Στο τέλος της μεθόδου αν κάποιο πεδίο βρίσκεται μέσα στη λίστα των περιορισμών τότε στην αντίστοιχη θέση ενός Boolean πίνακα ορίζουμε την τιμή **true**, δηλαδή ότι έχουμε ήδη ορίσει κάποιο περιορισμό για το συγκεκριμένο πεδίο.

```
1. private void showReflectFields(List<ReflectField>
   reflectFields) {
2.    reflectFieldsRecyclerViewAdapter = new
   ReflectFieldsRecyclerViewAdapter(reflectFields, userClasses,
   new OnListItemLongClickedListener() {
3.        @Override
4.        public void onListItemLongClicked(final ReflectField
   reflectField) {
5.            if (reflectField.getFieldType().isCollection() ||
   reflectField.getFieldType().isArray()) {
6.                Toast.makeText(ConstraintsActivity.this,
   getString(R.string.ConstraintsCollectionMSG),
   Toast.LENGTH_LONG).show();
7.            } else if
   (userClasses.contains(reflectField.getFieldType().getName()))
   {
8.                Intent intent = new
   Intent(ConstraintsActivity.this, ConstraintsActivity.class);
9.                intent.putExtra(getString(R.string.CLASS_NAME),
   reflectField.getFieldType().getName());
10.                   if (classPath != null) {
11.
   intent.putExtra(getString(R.string.CLASS_PATH), classPath +
   getString(R.string.Dot) + reflectField.getName());
12.                   } else {
13.
   intent.putExtra(getString(R.string.CLASS_PATH),
   reflectClassName + getString(R.string.Dot) +
   reflectField.getName());
14.                   }
15.                   try {
16.                       ConstraintsJsonData constraintsJsonData =
   new ConstraintsJsonData();
17.
   constraintsJsonData.setConstraints(myConstraints);
18.
   intent.putExtra(getString(R.string.ConsJSOND),
   mapper.writeValueAsString(constraintsJsonData));
19.                   } catch (JsonProcessingException e) {
20.                       e.printStackTrace();
21.                   }
22.                   startActivityForResult(intent,
   Constants.REQUEST_CODE);
23.               }
```

Εικόνα 67 "Μέθοδος showReflectFields (Α' Μέρος)"

```
1. else {
2.           ConstraintDialogFragment constraintDialogFragment =
   ConstraintDialogFragment.newInstance(reflectField.getName(),
3.                   reflectField.getFieldType().getName(),
   reflectClassName, new
   ConstraintDialogFragment.OnSaveButtonClickedListener() {
4.                       @Override
5.                       public void onSaveButtonClicked(String
   value, int operator) {
6.
   reflectFieldsRecyclerViewAdapter.setHasConstraint(reflectField
   , true);
7.
   reflectFieldsRecyclerViewAdapter.notifyDataSetChanged();
8.                           MyConstraint myConstraint = new
   MyConstraint();
9.                           List<String> path;
10.                             if (classPath != null) {
11.                                 path = new
   ArrayList<>(Arrays.asList(classPath.split("\\.")));
12.                             } else {
13.                                 path = new ArrayList<>();
14.                                 path.add(reflectClassName);
15.                             }
16.                             path.add(reflectField.getName());
17.                             myConstraint.setPath(path);
18.
   myConstraint.setOperator(operator);
19.
   myConstraint.setReflectFieldType(reflectField.getFieldType().g
   etName());
20.                             myConstraint.setValue(value);
21.                             myConstraints.add(myConstraint);
22.
   Toast.makeText(ConstraintsActivity.this,
   getString(R.string.ConstraintAdded) + reflectField.getName(),
   Toast.LENGTH_LONG).show();
23.                         }
24.                     });
25.   constraintDialogFragment.show(getSupportFragmentManager(),
   getString(R.string.constraintDialog));
26.         }
```

Εικόνα 68 "Μέθοδος showReflectFields (Β' Μέρος)"

```
1. for (ReflectField reflectField : reflectFields) {
2.         String reflectFieldName = reflectField.getName();
3.         for (MyConstraint myConstraint : myConstraints) {
4.             List<String> path = myConstraint.getPath();
5.             if (classPath != null) {
6.                 String[] splitClassPath =
   classPath.split("\\.");
7.                 if (path.get(path.size() -
   2).equals(splitClassPath[splitClassPath.length - 1]) &&
   path.get(path.size() - 1).equals(reflectFieldName)) {
8.
   reflectFieldsRecyclerViewAdapter.setHasConstraint(reflectFiel
   d, true);
9.                 }
10.             }
11.         }
12.     }
13.
   reflectFieldsRecyclerView.setAdapter(reflectFieldsRecyclerVie
   wAdapter);
14.   }
```

Εικόνα 69 "Μέθοδος showReflectFields (Γ' Μέρος)"

Η εσωτερική κλάση **GetReflectFields**, κώδικας της Εικόνας 70, θα κάνει χρήση των μεθόδων πριν την εκτέλεση, κατά την εκτέλεση και μετά την εκτέλεση. Στην μέθοδο **"onPreExecute()"** αρχικοποιούμε τις μεταβλητές που θα χρησιμοποιήσουμε, τη λίστα τύπου ReflectFields **"reflectFields"**, και στη συνέχεια αφού αρχικοποιήσουμε το αντικείμενο τύπου ProgressDialog για χρήση στην συγκεκριμένη Activity θέτουμε το μήνυμα που θα εμφανίζεται και τέλος το παρουσιάζουμε στο προσκήνιο. Στην μέθοδο **"doInBackground()"** εκτελείται η σύνδεση με τη Βάση, η λήψη των αποτελεσμάτων και ο τερματισμός της σύνδεσης. Αυτό συμβαίνει με τη χρήση της κλάσης Db4oSubClass, αρχικά δημιουργούμε ένα αντικείμενο τύπου **Db4oSubClass** περνώντας παραμετρικά τη μεταβλητή τύπου **Context** για τη διαχείριση των SharedPreferences από την Db4oSubClass. Μετά με τη χρήση της μεθόδου **"reflectFieldsNameASRF()"** και περνώντας παραμετρικά το όνομα της επιλεγμένης κλάσης ως αλφαριθμητικό λαμβάνουμε τα πεδία της κλάσης αυτής σε μορφή λίστας **ReflectFields**. Στη συνέχεια με τη χρήση της μεθόδου **"reflectClassesAsSTR()"** λαμβάνουμε τα ονόματα των κλάσεων που βρίσκονται αποθηκευμένες στη Βάση Δεδομένων και έχουν δημιουργηθεί από τον χρήστη σε μορφή λίστας τύπου αλφαριθμητικό. Με τη βοήθεια της **"CloseDB()"** τερματίζουμε τη σύνδεση με τον Εξυπηρετητή. Στη συνέχεια με τη μέθοδο **"onPostExecute()"** τερματίζουμε και επαναφέρουμε στο παρασκήνιο το αντικείμενο τύπου ProgressDialog.

```
1.    class GetReflectFields extends AsyncTask<String, Void,
   Void> {
2.        ProgressDialog mProgressDialog;
3.        List<ReflectField> reflectFields;
4.        protected void onPreExecute() {
5.            super.onPreExecute();
6.            reflectFields = new ArrayList<>();
7.            mProgressDialog = new
   ProgressDialog(ConstraintsActivity.this);
8.            mProgressDialog.setIndeterminate(true);
9.            mProgressDialog.setTitle("Loading Fields");
10.               mProgressDialog.setMessage("Searching for the
   required Fields");
11.               mProgressDialog.show();
12.           }
13.           protected Void doInBackground(String... params) {
14.               Db4oSubClass db4oSubClass = new
   Db4oSubClass(ctx);
15.               reflectFields =
   db4oSubClass.reflectFieldsNameASRF(params[0]);
16.               userClasses = db4oSubClass.reflectClassesAsSTR();
17.               db4oSubClass.CloseDB();
18.               return null;
19.           }
20.           protected void onPostExecute(Void aVoid) {
21.               super.onPostExecute(aVoid);
22.               showReflectFields(reflectFields);
23.               mProgressDialog.dismiss();
24.           }
25.       }
26.   }
```

Εικόνα 70 "Κλάση GetReflectFields"

3.2.5 ConstraintDialogFragment.java

Η κλάση **"ConstraintDialogFragment"**, κώδικας της Εικόνας 71, κληρονομεί την κλάση **"DialogFragment"** και η λειτουργικότητα της κλάσης αφορά την δημιουργία και την διαχείριση ενός παραθύρου το οποίο θα εμφανίζεται πάνω από το Γραφικό Περιβάλλον της κλάσης **"ConstraintsActivity"** χωρίς ωστόσο να την τοποθετεί στις Activity παρασκηνίου, βασική λειτουργία των DialogFragments είναι η προβολή μηνυμάτων, επιβεβαίωση ενεργειών καθώς και η εισαγωγή δεδομένων από τον χρήστη. Στις γραμμές δύο έως τέσσερα ορίζουμε τις τοπικές σταθερές που αφορούν τα ονόματα των παραμέτρων που θα περνάμε παραμετρικά στην **"ConstraintsActivity"** και πιο συγκεκριμένα για το όνομα του πεδίου, τον τύπο δεδομένων του πεδίου καθώς και το όνομα της κλάσης στην οποία ανήκει. Στη συνέχεια δηλώνουμε τις global μεταβλητές που θα χρησιμοποιήσουμε και με τη σειρά που εμφανίζονται είναι οι εξής, η μεταβλητή **"reflectFieldName"** τύπου αλφαριθμητικό το οποίο θα χρειαστούμε για να πάρουμε όλες τις τιμές του

συγκεκριμένου πεδίου στις περιπτώσεις που χρησιμοποιείται κάποιο αντικείμενο τύπου Spinner για την προβολή αυτών. Το πεδίο **“reflectFieldType”** τύπου αλφαριθμητικό θα το χρησιμοποιήσουμε για να θέσουμε τα κατάλληλα operator ανάλογα με τον τύπο δεδομένων του εκάστοτε πεδίου, αλφαριθμητικό, αριθμητικό, κλπ. Το πεδίο **“operatorSpinner”** τύπου Spinner χρησιμοποιείται για την προβολή των κατάλληλων operator ανάλογα με τον τύπο δεδομένων του εκάστοτε πεδίου. Στις γραμμές εννέα και δέκα ανάλογα με τον τύπο δεδομένων της εκάστοτε μεταβλητής, για παράδειγμα αν είναι η μεταβλητή είναι τύπου αλφαριθμητικού και ο χρήστης έχει επιλέξει τον operator **“Equals”** τότε θα εμφανιστούν στο αντικείμενο τύπου **Spinner** όλες οι τιμές του πεδίου αυτού, σε περίπτωση που επιλεγεί διαφορετικός τύπος δεδομένων ή διαφορετικός operator τότε θα χρησιμοποιήσουμε ένα αντικείμενο τύπου **EditText** όπου και ο χρήστης θα εισάγει την τιμή που επιθυμεί. Στην γραμμή έντεκα έχουμε τη μεταβλητή **“valueTextInputLayout”** τύπου TextInputLayout που θα χρησιμοποιηθεί για να θέσουμε τον τρόπο προβολής του πληκτρολογίου, σε περίπτωση αριθμού το πληκτρολόγιο θα διαθέτει μόνο αριθμούς και όχι αλφαριθμητικά. Η μεταβλητή τύπου Boolean **“isBoolean”** ορίζεται ως αληθής ή ψευδής ανάλογα με το αν το εκάστοτε αντικείμενο είναι τύπου Boolean ή όχι. Επειδή θα θέλαμε να προστατέψουμε τον χρήστη από το να θέσει περιορισμό δίχως να έχει ορίσει πρώτα operator και μια τιμή για σύγκριση θα υπερκαλύψουμε τη λειτουργικότητα του κουμπιού “αποθήκευση”, για να επιτευχθεί αυτό θα πρέπει να χρησιμοποιήσουμε τη μεταβλητή **“mOnSaveButtonClickedListener”** τύπου onSaveButtonClickedListener. Η μεταβλητή **“ctx”** τύπου Context θα χρησιμοποιηθεί για να περάσουμε παραμετρικά ένα instance του DialogFragment στην κλάση **Db4oSubClass** για την διαχείριση του αντικειμένου Sharedpreferences. Τέλος, η μεταβλητή **“fieldValues”** τύπου αλφαριθμητικό πίνακα χρησιμοποιείται για την αποθήκευση των τιμών που θα προβληθούν μέσω του **Spinner** που είναι υπεύθυνος για την προβολή τιμών στην περίπτωση που έχουμε μεταβλητή τύπου αλφαριθμητικό και χρησιμοποιούμε τον operator **“Equals”**. Στη συνέχεια, κώδικας της Εικόνας 72, ορίζουμε τους στατικούς αλφαριθμητικούς πίνακες που περιέχουν τα operator που θα φορτωθούν στο τύπου Spinner που είναι υπεύθυνο για την εμφάνισή τους.

```
1. public class ConstraintDialogFragment extends DialogFragment
   {
2.
3.    private static final String ARG_REFLECT_FIELD_NAME =
   "reflectFieldName";
4.    private static final String ARG_REFLECT_FIELD_TYPE =
   "reflectFieldType";
5.    private static final String ARG_REFLECT_CLASS_NAME =
   "reflectClassName";
6.
7.    private String reflectFieldName;
8.    private String reflectFieldType;
9.    private String reflectClassName;
10.      private Spinner operatorSpinner;
11.      private Spinner valueSpinner;
12.      private EditText valueEditText;
13.      private TextInputLayout valueTextInputLayout;
14.      private boolean isBoolean = false;
15.      private static OnSaveButtonClickedListener
   mOnSaveButtonClickedListener;
16.      private AlertDialog mAlertDialog;
17.      private Context ctx;
18.      private String[] fieldValues;
```

Εικόνα 71 "Κλάση ConstraintDialogFragment"

```
1. private static final String[] numbersOperators = new
   String[6];
2. static {
3.     numbersOperators[0] = "Select operator";
4.     numbersOperators[1] = "=";
5.     numbersOperators[2] = ">";
6.     numbersOperators[3] = "<";
7.     numbersOperators[4] = ">=";
8.     numbersOperators[5] = "<=";
9. }
10.   private static final String[] stringOperators = new
   String[3];
11.   static {
12.       stringOperators[0] = "Select operator";
13.       stringOperators[1] = "(=)/Equals";
14.       stringOperators[2] = "Contains";
15.   }
16.   private static final String[] charOrBooleanOperators = new
   String[2];
17.   static {
18.       charOrBooleanOperators[0] = "Select operator";
19.       charOrBooleanOperators[1] = "=";
20.   }
```

Εικόνα 72 "Στατικοί πίνακες για τα αντικείμενα τύπου Spinner"

Στη συνέχεια, κώδικας της Εικόνας 73, έχουμε τις μεθόδους **"public static ConstraintDialogFragment newInstance(String reflectFieldName, String reflectFieldType, String reflectClassName, OnSaveButtonClickedListener listener)"**, **"public void onCreate(Bundle savedInstanceState)"** και **"public Dialog onCreateDialog(Bundle savedInstanceState)"**. Η πρώτη μέθοδος χρησιμοποιείται όταν η Activity αναδημιουργείται, δηλαδή σε περιπτώσεις όπου γίνεται αλλαγή από Portrait σε Landscape mode. Στην περίπτωση που εκτελεστεί η μέθοδος αυτή τότε θα αποθηκευτούν τα αντίστοιχα πεδία, όνομα πεδίου, όνομα κλάσης στην οποία βρίσκεται και ο τύπος δεδομένων του πεδίου, με τη χρήση αντικειμένου Bundle, παρόμοιο με αυτό που είδαμε σε άλλες κλάσεις το Intent. Η μέθοδος **"onCreate()"** δεν διαθέτει κάποια λειτουργικότητα καθώς με βάση τις οδηγίες που μας προσφέρει η Google για το Λειτουργικό Σύστημα Android στην περίπτωση που θέλουμε να υλοποιήσουμε ένα DialogFragment με δικά μας χαρακτηριστικά, custom DialogFragment, θα πρέπει να χρησιμοποιήσουμε την μέθοδο **"onCreateDialog()"** αντί της **"onCreate()"**. Στις πρώτες επτά σειρές της **"onCreateDialog()"** αποθηκεύουμε ένα instance του DialogFragment για χρήση του αντικειμένου SharedPreferences στην κλάση Db4oSubClass και ανακτούμε τα πεδία τύπος δεδομένων του πεδίου, το όνομα του πεδίου και το όνομα της κλάσης στην οποία ανήκει το πεδίο. Στα κομμάτια κώδικα των Εικόνων 74 και 75,

δημιουργούμε το custom Alert Dialog όπου ο χρήστης θα θέσει τους περιορισμούς. Έπειτα ορίζουμε τι θα συμβεί στην περίπτωση που ο χρήστης πατήσει το κουμπί “αποθήκευση”, όπου ανάλογα με τον επιλεγμένο operator αποθηκεύουμε τον αντίστοιχο κωδικό σύμφωνα με την κλάση **“Constants”**. Ανάλογα με το αν έχουμε χρησιμοποιήσει αντικείμενο τύπου Spinner ή EditText για τη λήψη της τιμής περιορισμού που έχει θέσει ο χρήστης αποθηκεύουμε την τιμή του ανάλογου αντικειμένου. Στη περίπτωση που ο χρήστης πατήσει το κουμπί “ακύρωση” τότε θα κλείσει το DialogFragment χωρίς να γίνει κάποια αποθήκευση τιμών. Στη συνέχεια της μεθόδου, **“onCreateDialog()”**, κώδικας της Εικόνας 76, αφού κάνουμε τη σύνδεση μεταξύ των στοιχείων Γραφικού Περιβάλλοντος με τις μεταβλητές που χρησιμοποιούμε για τον προγραμματιστικό έλεγχο αυτών, ορίζουμε τι θα συμβεί στην περίπτωση που χρησιμοποιήσουμε το αντικείμενο τύπου EditText στην περίπτωση που γίνει αλλαγή στην τιμή του πεδίου αυτού, **”public void onTextChanged(CharSequence s, int start, int before, int count)”**, που είναι η εκτέλεση της μεθόδου **“shouldPositiveButtonEnabled()”**. Οι υπόλοιπες μέθοδοι **“public void beforeTextChanged(CharSequence s, int start, int count, int after)”** και **“public void afterTextChanged(Editable s)”** δεν χρησιμοποιούνται αλλά η ύπαρξη τους είναι επιβεβλημένη καθώς πρέπει να υλοποιηθούν σύμφωνα με το EditText.addTextChangedListener. Στην περίπτωση που χρησιμοποιήσουμε αντικείμενο τύπου Spinner θέτουμε ότι στην περίπτωση που επιλεχθεί κάποιο αντικείμενο από τη λίστα αυτού τότε θα εκτελεστεί η μέθοδος **“shouldPositiveButtonEnabled()”**. Στις γραμμές ένα έως είκοσι πέντε, κώδικας της Εικόνας 77, δημιουργούμε έναν απλό τύπου αλφαριθμητικό **Adapter** όπου ανάλογα με τον τύπο του επιλεγμένου πεδίου φορτώνουμε τον αντίστοιχο πίνακα operator που είδαμε νωρίτερα και καθορίζουμε τον τύπο εισαγωγής των δεδομένων. Στη συνέχεια συνδέουμε τον **Adapter** με το Γραφικό Περιβάλλον. Στις γραμμές ένα έως τριάντα, κώδικας της Εικόνας 78, δηλώνουμε τί θα συμβεί αν ο χρήστης επιλέξει κάποιο αντικείμενο από το αντικείμενο τύπου Spinner. Αν πρόκειται για αλφαριθμητικό με τον operator **“Equals”** τότε το αντικείμενο τύπου EditText αφαιρείται από το προσκήνιο ενώ ταυτόχρονα το αντικείμενο τύπου Spinner έρχεται στο προσκήνιο και εκτελείται η κλάση **LoadFieldvalues**, που αποτελεί την κλάση που εκτελεί το Thread για την επιστροφή των τιμών του πεδίου στον πίνακα **“fieldValues”** και πραγματοποιείται η εμφάνιση αυτών μέσα στο Γραφικό αντικείμενο. Στην περίπτωση που έχουμε αντικείμενο τύπου “Boolean” τότε το αντικείμενο τύπου EditText αφαιρείται από το προσκήνιο ενώ ταυτόχρονα το αντικείμενο τύπου Spinner έρχεται στο προσκήνιο και θέτουμε σε αυτό τις τιμές **“Αληθές”** και **“Ψευδές”**. Σε κάθε άλλη περίπτωση οδηγούμε στο παρασκήνιο το αντικείμενο τύπου Spinner και οδηγούμε το αντικείμενο τύπου EditText στο προσκήνιο έτσι ώστε ο χρήστης να εισάγει την τιμή για το επιλεγμένο περιορισμό που επιθυμεί. Τέλος, μεταβαίνουμε στη μέθοδο **“shouldPositiveButtonEnabled()”**. Η μέθοδος **“public void onNothingSelected(AdapterView<?> parent)”** δεν χρησιμοποιείται καθώς με την υπόλοιπη λειτουργικότητα που έχουμε καθορίσει στο Alert Dialog ο χρήστης

οδηγείται στο να επιλέξει κάποιο αντικείμενο από τη λίστα έτσι ώστε να προχωρήσει στον ορισμό ενός περιορισμού.

```
1. public static ConstraintDialogFragment newInstance(String
   reflectFieldName, String reflectFieldType, String
   reflectClassName, OnSaveButtonClickedListener listener) {
2.     mOnSaveButtonClickedListener = listener;
3.     ConstraintDialogFragment fragment = new
   ConstraintDialogFragment();
4.     Bundle args = new Bundle();
5.     args.putString(ARG_REFLECT_FIELD_NAME, reflectFieldName);
6.     args.putString(ARG_REFLECT_FIELD_TYPE, reflectFieldType);
7.     args.putString(ARG_REFLECT_CLASS_NAME, reflectClassName);
8.     fragment.setArguments(args);
9.     return fragment;
10.   }
11.   public void onCreate(Bundle savedInstanceState) {
12.       super.onCreate(savedInstanceState);
13.   }
14.   public Dialog onCreateDialog(Bundle savedInstanceState) {
15.       ctx = getActivity();
16.       if (getArguments() != null) {
17.           reflectFieldName =
   getArguments().getString(ARG_REFLECT_FIELD_NAME);
18.           reflectFieldType =
   getArguments().getString(ARG_REFLECT_FIELD_TYPE);
19.           reflectClassName =
   getArguments().getString(ARG_REFLECT_CLASS_NAME);
20.       }
```

Εικόνα 73 "Μέθοδοι newInstance, onCreate και onCreateDialog (Α' Μέρος)"

```
1. AlertDialog.Builder builder = new
   AlertDialog.Builder(getActivity());
2. LayoutInflater inflater = getActivity().getLayoutInflater();
3. View view = inflater.inflate(R.layout.dialog_constraint,
   ((ViewGroup) getView()));
4. builder.setTitle(getString(R.string.ConstraintFragmentTitle) +
   reflectFieldName)
5.         .setPositiveButton(getString(R.string.SaveButton), new
   DialogInterface.OnClickListener() {
6.             public void onClick(DialogInterface dialog, int id)
   {
7.                 if (mOnSaveButtonClickedListener != null) {
8.                     int operator;
9.                     String value;
10.                       switch
   (operatorSpinner.getSelectedItem().toString()) {
11.                           case "=":
12.                           case "(=)/Equals":
13.                               operator =
   Constants.EQUALS_OPERATOR;
14.                               break;
15.                           case ">":
16.                               operator =
   Constants.GREATER_OPERATOR;
17.                               break;
18.                           case "<":
19.                               operator =
   Constants.SMALLER_OPERATOR;
20.                               break;
21.                           case ">=":
22.                               operator =
   Constants.GREATER_EQUALS_OPERATOR;
23.                               break;
24.                           case "<=":
25.                               operator =
   Constants.SMALLER_EQUALS_OPERATOR;
26.                               break;
27.                           case "Like":
28.                               operator =
   Constants.LIKE_OPERATOR;
29.                               break;
30.                           default:
31.                               operator = -1;
32.                               break;
33.                       }
```

Εικόνα 74 "Μέθοδος onCreateDialog (Β' Μέρος)"

```
1.   if (valueTextInputLayout.getVisibility() == View.VISIBLE) {
2.                value = valueEditText.getText().toString();
3.            } else {
4.                value =
   valueSpinner.getSelectedItem().toString();
5.            }
6.
   mOnSaveButtonClickedListener.onSaveButtonClicked(value,
   operator);
7.        }
8.        dismiss();
9.    }
10.   })
11.   .setNegativeButton(getString(R.string.CancelButton), new
   DialogInterface.OnClickListener() {
12.       public void onClick(DialogInterface dialog, int id) {
13.           dismiss();
14.       }
15.   })
16.   .setView(view);
```

Εικόνα 75 "Μέθοδος onCreateDialog (Γ' Μέρος)"

```
1. operatorSpinner = (Spinner)
   view.findViewById(R.id.operatorSpinner);
2. valueSpinner = (Spinner)
   view.findViewById(R.id.valueSpinner);
3. valueEditText = (EditText)
   view.findViewById(R.id.valueEditText);
4. valueTextInputLayout = (TextInputLayout)
   view.findViewById(R.id.valueTextInputLayout);
5. valueEditText.addTextChangedListener(new TextWatcher() {
6.    public void beforeTextChanged(CharSequence s, int start,
   int count, int after) {
7.    }
8.    public void onTextChanged(CharSequence s, int start, int
   before, int count) {
9.        shouldPositiveButtonEnabled();
10.     }
11.     public void afterTextChanged(Editable s) {
12.     }
13.  });
14.  valueSpinner.setOnItemSelectedListener(new
   AdapterView.OnItemSelectedListener() {
15.     public void onItemSelected(AdapterView<?> parent, View
   view, int position, long id) {
16.         shouldPositiveButtonEnabled();
17.     }
18.     public void onNothingSelected(AdapterView<?> parent) {
19.     }
```

Εικόνα 76 "Μέθοδος onCreateDialog (Δ' Μέρος)"

```
1. ArrayAdapter adapter;
2. switch (reflectFieldType) {
3.     case ReflectMTypes.STRING:
4.         adapter = new ArrayAdapter<>(getActivity(),
   android.R.layout.simple_spinner_item, stringOperators);
5.
   valueEditText.setInputType(InputType.TYPE_TEXT_FLAG_NO_SUGGEST
   IONS);
6.         break;
7.     case ReflectMTypes.BOOLEAN:
8.         isBoolean = true;
9.         adapter = new ArrayAdapter<>(getActivity(),
   android.R.layout.simple_spinner_item, charOrBooleanOperators);
10.          break;
11.      case ReflectMTypes.CHAR:
12.          adapter = new ArrayAdapter<>(getActivity(),
   android.R.layout.simple_spinner_item, charOrBooleanOperators);
13.          break;
14.      default:
15.          adapter = new ArrayAdapter<>(getActivity(),
   android.R.layout.simple_spinner_item, numbersOperators);
16.          if (reflectFieldType.equals(ReflectMTypes.UTILDATE)
   || reflectFieldType.equals(ReflectMTypes.SQLDATE)) {
   valueEditText.setInputType(InputType.TYPE_CLASS_DATETIME);
17.          } else {
18.              int type = InputType.TYPE_CLASS_NUMBER |
   InputType.TYPE_NUMBER_FLAG_DECIMAL;
19.              valueEditText.setInputType(type);
20.          }
21.          break;
22.   }
23.   adapter.setDropDownViewResource(android.R.layout.simple_spin
   ner_dropdown_item);
```

Εικόνα 77 "Μέθοδος onCreateDialog (Ε' Μέρος)"

```
1. operatorSpinner.setOnItemSelectedListener(new
   AdapterView.OnItemSelectedListener() {
2.       public void onItemSelected(AdapterView<?> parent, View
   view, int position, long id) {
3.           String selectedOperator =
   parent.getItemAtPosition(position).toString();
4.           if
   ((selectedOperator.equals(getString(R.string.EqualsSign)) ||
   selectedOperator.equals(getString(R.string.Equals))) &&
   (reflectFieldType.equalsIgnoreCase(ReflectMTypes.STRING) ||
   reflectFieldType.equalsIgnoreCase(ReflectMTypes.CHAR))) {
5.
   valueTextInputLayout.setVisibility(View.INVISIBLE);
6.               valueSpinner.setVisibility(View.VISIBLE);
7.
8.               if (fieldValues == null) {
9.                   new LoadFieldValues().execute();
10.                } else {
11.                   ArrayAdapter adapter = new
   ArrayAdapter<>(getActivity(),
   android.R.layout.simple_spinner_item, fieldValues);
12.                   valueSpinner.setAdapter(adapter);
   adapter.setDropDownViewResource(android.R.layout.simple_spinne
   r_dropdown_item);
13.               }
14.           } else if (isBoolean) {
15.
   valueTextInputLayout.setVisibility(View.INVISIBLE);
16.               valueSpinner.setVisibility(View.VISIBLE);
17.               ArrayAdapter adapter = new
   ArrayAdapter<>(getActivity(),
   android.R.layout.simple_spinner_item, new
   String[]{getString(R.string.TRUE),
   getString(R.string.FALSE)});
18.               valueSpinner.setAdapter(adapter);
   adapter.setDropDownViewResource(android.R.layout.simple_spinne
   r_dropdown_item);
19.           } else {
20.
   valueTextInputLayout.setVisibility(View.VISIBLE);
21.               valueSpinner.setVisibility(View.INVISIBLE);
22.           }
23.           shouldPositiveButtonEnabled();
24.       }
25.       public void onNothingSelected(AdapterView<?> parent)
   { }  });
```

Εικόνα 78 "Μέθοδος onCreateDialog (ΣΤ' Μέρος)"

Στη συνέχεια, κώδικας της Εικόνας 79, έχουμε τις τρείς μεθόδους, την **"public void onStart()"**, την **"private void shouldPositiveButtonEnabled()"** και την **"public void onDetach()"**. Η μέθοδος **"onStart()"** εκτελείται την στιγμή που το εκάστοτε Fragment γίνεται ορατό στον χρήστη. Στη συγκεκριμένη περίπτωση θέτουμε το κουμπί "αποθήκευση" ως μη επιλέξιμο καθώς θέλουμε ο χρήστης να επιλέξει πρώτα operator και να θέσει τιμή σύγκρισης και στη συνέχεια να του δοθεί το δικαίωμα να αποθηκεύσει τον περιορισμό. Αυτό τον έλεγχο έχει αναλάβει να εκτελέσει η μέθοδος **"shouldPositiveButtonEnabled()"** η οποία αφού ελέγξει αν έχει επιλεγεί κάποιος operator και έχει οριστεί από τον χρήστη κάποια τιμή για την διενέργεια της σύγκρισης θέτει το κουμπί "αποθήκευση" ως επιλέξιμο. Τέλος, η μέθοδος **"onDetach()"** η οποία καλείται αμέσως μετά τη μέθοδο "onDestroy()",

που καλείται κατά τον τερματισμό του Fragment, απελευθερώνει τον Listener που είχαμε ορίσει για το κουμπί “αποθήκευση”.

```
1. public void onStart() {
2.     super.onStart();
3. mAlertDialog.getButton(AlertDialog.BUTTON_POSITIVE).setEnable
   d(false);
4. }
5. private void shouldPositiveButtonEnabled() {
6.     if (mAlertDialog != null) {
7.         Button positiveButton =
   mAlertDialog.getButton(AlertDialog.BUTTON_POSITIVE);
8.         if (((valueEditText.getText().length() > 0 &&
   valueTextInputLayout.getVisibility() == View.VISIBLE) ||
9.                 (valueSpinner.getSelectedItem() != null &&
   !valueSpinner.getSelectedItem().toString().equals(getString(R
   .string.Loading)) && valueSpinner.getVisibility() ==
   View.VISIBLE)) &&
10.                    operatorSpinner.getSelectedItemPosition()
   != 0) {
11.                positiveButton.setEnabled(true);
12.            } else {
13.                positiveButton.setEnabled(false);
14.            }
15.        }
16.    }
17.    public void onDetach() {
18.        super.onDetach();
19.        mOnSaveButtonClickedListener = null;
20.    }
```

Εικόνα 79 “Μέθοδοι onStart, shouldPositiveButtonEnabled και onDetached”

Στην εσωτερική κλάση **LoadFieldValues**, κώδικας της Εικόνας 80, θα κάνουμε χρήση των μεθόδων πριν την εκτέλεση, κατά την εκτέλεση και μετά την εκτέλεση. Στη μέθοδο **“onPreExecute()”** δημιουργούμε έναν Adapter απλού αλφαριθμητικού τύπου που ενημερώνει τον χρήστη ότι διενεργείται ένα ερώτημα στη Βάση Δεδομένων για τη λήψη όλων των τιμών του επιλεγμένου πεδίου τύπου αλφαριθμητικό και το εμφανίζουμε στο Γραφικό Περιβάλλον με τη βοήθεια του αντικειμένου **“valueSpinner”** τύπου Spinner.

```
1. private class LoadFieldValues extends AsyncTask<Void, Void,
   Void> {
2.
3.     @Override
4.     protected void onPreExecute() {
5.         super.onPreExecute();
6.         ArrayAdapter adapter = new
   ArrayAdapter<>(getActivity(),
   android.R.layout.simple_spinner_item, new
   String[]{getString(R.string.Loading)});
7.         valueSpinner.setAdapter(adapter);
8.
   adapter.setDropDownViewResource(android.R.layout.simple_spinn
   er_dropdown_item);
9.     }
```

Εικόνα 80 "Κλάση LoadFieldValues και Μέθοδος onPreExecute"

Στην μέθοδο **"doInBackground()"**, κώδικας της Εικόνας 81, εκτελείται η σύνδεση με τη Βάση, η λήψη των αποτελεσμάτων και ο τερματισμός της σύνδεσης. Αυτό συμβαίνει με τη χρήση της κλάσης Db4oSubClass, αρχικά δημιουργούμε ένα αντικείμενο τύπου **Db4oSubClass** περνώντας παραμετρικά τη μεταβλητή τύπου **Context** για τη διαχείριση των SharedPreferences από την Db4oSubClass. Μετά με τη χρήση της μεθόδου **"reflectFieldsNameASRF()"** και περνώντας παραμετρικά το όνομα της επιλεγμένης κλάσης ως αλφαριθμητικό λαμβάνουμε τα πεδία της κλάσης αυτής σε μορφή λίστας **ReflectFields**. Στη συνέχεια με τη χρήση της μεθόδου **"reflectClass()"** λαμβάνουμε ένα instance της κλάσης που έχουμε περάσει παραμετρικά ως αλφαριθμητικό, μετά λαμβάνουμε ένα instance του πεδίου που θέλουμε να θέσουμε τον περιορισμό περνώντας παραμετρικά στη μέθοδο **"getDeclaredField()"** το όνομα του πεδίου ως αλφαριθμητικό. Αφού ολοκληρώσουμε τις παραπάνω ενέργειες δημιουργούμε ένα ερώτημα (query) προς τη Βάση Δεδομένων θέτοντας ως περιορισμό το όνομα της κλάσης στην οποία βρίσκεται το πεδίο και το εκτελούμε. Στη συνέχεια αποθηκεύουμε στον πίνακα **"fieldValues"** όλες τις τιμές που πέρνει το πεδίο πάνω στο οποίο χτίζουμε τον περιορισμό. Με τη βοήθεια της μεθόδου **"CloseDB()"** τερματίζουμε τη σύνδεση με τον Εξυπηρετητή. Στη συνέχεια, με τη μέθοδο **"onPostExecute()"** δημιουργούμε έναν **Adapter** απλού αλφαριθμητικού τύπου που περιέχει όλες τις τιμές του επιλεγμένου πεδίου έτσι ώστε να επιλέξει ο χρήστης μία από αυτές.

```
1. protected Void doInBackground(Void... params) {
2.         Db4oSubClass db4oSubClass = new Db4oSubClass(ctx);
3.         ReflectClass reflectClass =
   db4oSubClass.reflectClass(reflectClassName);
4.         ReflectField reflectField =
   reflectClass.getDeclaredField(reflectFieldName);
5.         Query query = db4oSubClass.getDb().query();
6.         query.constrain(reflectClass);
7.         ObjectSet result = query.execute();
8.         fieldValues = new String[result.size()];
9.         for (int i = 0; i < result.size(); i++) {
10.              fieldValues[i] =
   reflectField.get(result.get(i)).toString();
11.          }
12.          db4oSubClass.CloseDB();
13.          return null;
14.      }
15.      protected void onPostExecute(Void aVoid) {
16.          super.onPostExecute(aVoid);
17.          ArrayAdapter adapter = new
   ArrayAdapter<>(getActivity(),
   android.R.layout.simple_spinner_item, fieldValues);
18.          valueSpinner.setAdapter(adapter);
19.   adapter.setDropDownViewResource(android.R.layout.simple_sp
   inner_dropdown_item);
20.      }
21.  }
```

Εικόνα 81 "Μέθοδοι doInBackground και onPostExecute"

3.2.6 RecursivePrint.java

Η κλάση **"RecursivePrint"** αποτελεί την Activity που είναι επιφορτισμένη με την παρουσίαση των αποτελεσμάτων του query, κώδικας της Εικόνας 82. Στις γραμμές δύο έως δώδεκα ορίζουμε τις global μεταβλητές που θα χρησιμοποιήσουμε. Με τη σειρά που εμφανίζονται αυτές είναι η μεταβλητή **"recursiveRecyclerView"** τύπου RecyclerView. Η μεταβλητή **"classPath"** τύπου αλφαριθμητικό έχει την διαδρομή (path) έως αυτή την κλάση. Για παράδειγμα αν ο χρήστης έχει επιλέξει στην κλάση Initial την κλάση Participates και στη συνέχεια το πεδίο athlete και στη συνέχεια το πεδίο trainer, που αποτελεί αναφορά σε αντικείμενο τύπου Trainer, τότε η τιμή της μεταβλητής classPath θα είναι Participates.athlete.trainer. Η λίστα αλφαριθμητικό **"userClasses"** θα φιλοξενήσει τα ονόματα όλων των κλάσεων που είναι αποθηκευμένα στην Βάση Δεδομένων και έχουν δημιουργηθεί από τον χρήστη. Η μεταβλητή **"mapper"** τύπου ObjectMapper θα χρησιμοποιηθεί για την μετατροπή αντικειμένων Java σε αντικείμενα JSON και αντίστροφα. Στην μεταβλητή **"constraintsJsonData"** τύπου ConstraintsJsonData, όπως είδαμε στην

παράγραφο 3.1.5, θα αποθηκεύσουμε τους περιορισμούς καθώς και τον operator (λογικό “και” ή λογικό “ή”) που ο χρήστης έθεσε στην κλάση ConstraintsActivity. Όταν έχουμε να προβάλλουμε μία λίστα από αντικείμενα του ίδιου είδους και ο χρήστης καλείται να διαλέξει ένα από αυτά τότε αποθηκεύουμε στην μεταβλητή τύπου ακέραιος αριθμός **“reflectClassIndex”** τη θέση του αντικειμένου στην αρχική λίστα των αποτελεσμάτων. Στο αλφαριθμητικό **“attributePath”** αποθηκεύουμε τη διαδρομή (path) από την αρχική μεταβλητή έως τη μεταβλητή την οποία έχουμε ανοίξει αυτή τη στιγμή, για παράδειγμα αν είμαστε στην κλάση Participates και φτάσουμε να βλέπουμε τα πεδία της κλάσης Trainer, τότε η μεταβλητή αυτή θα έχει τιμή athlete.trainer. Η αλφαριθμητική μεταβλητή **“className”** αποθηκεύει το όνομα της εκάστοτε κλάσης στην οποία βρισκόμαστε. Η μεταβλητή τύπου Boolean **“QuerFlag”** μας βοηθάει να γνωρίζουμε αν θα εμφανίσουμε λίστα αντικειμένων του ίδιου είδους ή όχι. Τέλος η μεταβλητή **“ctx”** τύπου Context, θα χρησιμοποιηθεί για να περάσουμε παραμετρικά στην κλάση Db4oSubClass ένα instance της Activity ConstraintsActivity για να είναι δυνατή η προσπέλαση των SharedPreferences από την κλάση Db4oSubClass.

```
1. public class RecursivePrint extends AppCompatActivity {
2.
3.     private RecyclerView recursiveRecyclerView;
4.     private ReflectFieldsValuesRecyclerViewAdapter
   reflectFieldsValuesRecyclerViewAdapter;
5.     private String classPath;
6.     private List<String> userClasses;
7.     private static ObjectMapper mapper = new ObjectMapper();
8.     private ConstraintsJsonData constraintsJsonData;
9.     private int reflectClassIndex;
10.        private String attributePath;
11.        private String className;
12.        private boolean QueryFlag;
13.        private Context ctx;
```

Εικόνα 82 “Κλάση RecursivePrint”

Στον κώδικα της Εικόνας 83, στις γραμμές τέσσερα και πέντε αφού συνδέσουμε την επιθυμητή γραμμή εργαλείων με την αντίστοιχη μεταβλητή του προγράμματός μας, την ορίζουμε ως την ενεργή γραμμή εργαλείων της Activity μας. Στη γραμμή έξι αποθηκεύουμε ένα instance της Activity στην μεταβλητή τύπου Context για χρήση αυτής στην κλάση Db4oSubClass. Έπειτα, ορίζουμε το layout του αντικειμένου RecycleView. Στη γραμμή ένα, κώδικας της Εικόνας 84, ανακτούμε την τιμή του πεδίου **“className”** που έχουμε περάσει παραμετρικά από την Activity που εκκίνησε αυτή την Activity και είναι τύπου αλφαριθμητικό. Στις γραμμές

δύο έως πέντε, ανακτούμε την τιμή του πεδίου **“classPath”** που έχουμε περάσει παραμετρικά από την Activity που εκκίνησε αυτή την Activity και είναι τύπου αλφαριθμητικό. Στη συνέχεια ελέγχουμε ότι η μεταβλητή **“classPath”** περιέχει κάποια τιμή και σε περίπτωση που είναι κενή θέτουμε ως τιμή της μεταβλητής αυτής την τιμή της μεταβλητής **“className”**. Στις γραμμές έξι μέχρι δεκατρία ανακτούμε την τιμή του πεδίου τύπου **ConstraintsJsonData** που έχουμε περάσει παραμετρικά από την Activity που εκκίνησε αυτή την Activity και είναι τύπου αλφαριθμητικό. Στη συνέχεια ελέγχουμε ότι η μεταβλητή **“jsonData”** περιέχει κάποια κάποια τιμή και με τη χρήση του αντικειμένου **ObjectMapper** αποσειριοποιούμε τα δεδομένα και μεταφράζουμε το JSON αρχείο σε αντικείμενο τύπου **ConstraintsJsonData**. Έπειτα ανακτούμε την τιμή του πεδίου **“attributePath”** που έχουμε περάσει παραμετρικά από την Activity που εκκίνησε αυτή την Activity και είναι τύπου αλφαριθμητικό. Το πεδίο **“reflectClassIndex”** αφορά την θέση του αντικειμένου που επιλέξαμε στην αρχική λίστα αποτελεσμάτων του ερωτήματος (query), επειδή είναι τύπου ακέραιος αριθμός χρησιμοποιούμε τη μέθοδο **“getIntent().getIntExtra()”** με προκαθορισμένη τιμή σε περίπτωση που δεν έχει οριστεί, για να την ανακτήσουμε αφού την έχουμε περάσει παραμετρικά από την Activity που εκκίνησε αυτή την Activity. Η μεταβλητή **“QueryFlag”** τύπου Boolean αφορά το γεγονός αν θα εμφανίσουμε μια λίστα με αντικείμενα του ίδιου τύπου ή όχι και την ανακτούμε με τη μέθοδο **“getintent().getBooleanExtra()”** με προκαθορισμένη τιμή σε περίπτωση που δεν έχει οριστεί, για να την ανακτήσουμε αφού την έχουμε περάσει παραμετρικά από την Activity που εκκίνησε αυτή την Activity. Στο τέλος της μεθόδου αυτής καλούμε και εκτελούμε την κλάση **RunQuery** περνώντας ώς παράμετρο το όνομα της κλάσης για να εκτελέσουμε σε Thread το ερώτημα (query).

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_recursive_print);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    ctx = this;
    recursiveRecyclerView = (RecyclerView)
findViewById(R.id.recursiveprintRecyclerView);
    recursiveRecyclerView.setLayoutManager(new
LinearLayoutManager(this));
    recursiveRecyclerView.addItemDecoration(new
DividerItemDecoration(this));
      List<String> emptyList = new ArrayList<>();
      reflectFieldsValuesRecyclerViewAdapter = new
ReflectFieldsValuesRecyclerViewAdapter(emptyList, null,
null);

recursiveRecyclerView.setAdapter(reflectFieldsValuesRecyclerV
iewAdapter);
```

Εικόνα 83 "Μέθοδος onCreate (Α' Μέρος)"

```
className = getIntent().getStringExtra("className");
        classPath = getIntent().getStringExtra("classPath");
        if (classPath == null) {
            classPath = className;
        }
        String jsonData =
   getIntent().getStringExtra("ConstraintsJsonData");
        if (jsonData != null) {
            try {
                constraintsJsonData =
   mapper.readValue(jsonData, ConstraintsJsonData.class);
              } catch (IOException e) {
                  e.printStackTrace();
              }
          }
          attributePath =
   getIntent().getStringExtra("attributePath");
          reflectClassIndex =
   getIntent().getIntExtra("reflectClassIndex", -1);
          QueryFlag =
   getIntent().getBooleanExtra("QueryFlag", false);
          new RunQuery().execute(className);
          ActionBar actionBar = getSupportActionBar();
          if (actionBar != null) {
              actionBar.setDisplayHomeAsUpEnabled(true);
          }
```

Εικόνα 84 "Μέθοδος onCreate (Β' Μέρος)"

Η μέθοδος **“private Object correctTypeConverter(Object value, Object type)”**, κώδικας της Εικόνας 85, έχει ως σκοπό να διορθώσει το πρόβλημα που προκύπτει στην εκτέλεση του ερωτήματος (query) καθώς όλες οι τιμές των πεδίων που ορίζονται στους περιορισμούς είναι τύπου αλφαριθμητικό, ανεξάρτητα αν αυτές στην πραγματικότητα είναι άλλου τύπου δεδομένων. Ως είσοδο δέχεται την τιμή που έχει θέσει ο χρήστης για τον περιορισμό και τον τύπο δεδομένων της μεταβλητής αυτής. Με ένα **switch-case** statement ανάλογα με τον τύπο της μεταβλητής και τη χρήση της κλάσης **ReflectMTypes** που είδαμε στην παράγραφο 3.1.2 κάνουμε type Casting στο σωστό τύπο δεδομένων. Οι τύποι δεδομένων που υποστηρίζει αυτή η μέθοδος είναι οι βασικοί τύποι δεδομένων της γλώσσας προγραμματισμού Java. Σε περίπτωση που δεν γίνει κάποια ταύτιση τότε η τιμή είναι όπως πριν εκτελεστεί αυτή η μέθοδος και στη συνέχεια ανεξαρτήτως αποτελέσματος γίνεται η επιστροφή της τιμής του πεδίου είτε διορθωμένη είτε όχι.

```
1. private Object correctTypeConverter(Object value, Object type)
   {
2.     Object temp = null;
3.     switch (type.toString()) {
4.         case ReflectMTypes.FLOAT:
5.             temp = Float.parseFloat(value.toString());
6.             break;
7.         case ReflectMTypes.INT:
8.             temp = Integer.parseInt(value.toString());
9.             break;
10.          case ReflectMTypes.BOOLEAN:
11.              temp = Boolean.parseBoolean(value.toString());
12.              break;
13.          case ReflectMTypes.BYTE:
14.              temp = Byte.parseByte(value.toString());
15.              break;
16.          case ReflectMTypes.DOUBLE:
17.              temp = Double.parseDouble(value.toString());
18.              break;
19.          case ReflectMTypes.LONG:
20.              temp = Long.parseLong(value.toString());
21.              break;
22.          case ReflectMTypes.SHORT:
23.              temp = Short.parseShort(value.toString());
24.              break;
25.          default:
26.              temp = value;
27.      }
28.      return temp;
29.  }
```

Εικόνα 85 “Μέθοδος correctTypeConverter”

Η μέθοδος **“public Constraint MyQ(List<Object> s, Query q, int operator)”**, κώδικας της Εικόνας 86, είναι μια αναδρομική μέθοδος η οποία δέχεται ως είσοδο μία λίστα τύπου αντικείμενο (Object), μία μεταβλητή τύπου **Query** και μία μεταβλητή τύπου ακέραιος αριθμός που είναι ο κωδικός του operator, όπως αυτός έχει δηλωθεί στην κλάση **Constants** που είδαμε στην παράγραφο 3.1.3, και καλείται για να λύσει το πρόβλημα του πως θα χτίσουμε έναν περιορισμό έτσι ώστε να τον εκτελέσουμε. Στην αρχή της μεθόδου ελέγχουμε αν η λίστα που έχουμε περάσει παραμετρικά έχει δύο αντικείμενα εντός της, δηλαδή την τιμή που θέλουμε να ορίσουμε ως περιορισμό και τον τύπο της μεταβλητής αυτής. Στην συνέχεια έχουμε ένα **switch-case** statement όπου με τη χρήση της κλάσης **Constants** χτίζουμε το τελευταίο κομμάτι του ερωτήματος (query). Αξίζει να αναφέρουμε ότι επειδή δεν υπάρχει native υποστήριξη από την db4o για τις περιπτώσεις μικρότερο ίσο και μεγαλύτερο ίσο, σε αυτές τις περιπτώσεις οδηγηθήκαμε στο να εκτελέσουμε άλλη μια φορά τη μέθοδο **“MyQ()”** καλώντας τις μεθόδους μεγαλύτερο και μικρότερο αντίστοιχα και στη συνέχεια τις συναλιθεύσουμε με τη χρήση λογικού “ή” με την εκτέλεση της **“MyQ”** για τον operator ίσον. Στην περίπτωση που η λίστα που έχουμε περάσει παραμετρικά στη μέθοδο έχει περισσότερα από δύο αντικείμενα τότε δημιουργούμε ένα δεύτερο **Query** και ορίζουμε ότι είναι το Query που περάσαμε παραμετρικά στη μέθοδο **“MyQ”** με περιορισμό πεδίου, όπως βλέπουμε στη γραμμή δεκαοχτώ, τοποθετώντας το πρώτο αντικείμενο που βρίσκεται στη λίστα. Στη συνέχεια αφαιρούμε το πρώτο αντικείμενο από την παλιά λίστα και καλούμε την μέθοδο **“MyQ”**, αναδρομή, με παραμέτρους την λίστα που δέχτηκε ως είσοδο μειωμένη σε μέγεθος κατά ένα αντικείμενο, το νέο αντικείμενο τύπου Query που δημιουργήσαμε και τέλος την μεταβλητή operator που δέχτηκε ως είσοδο.

```
1. public Constraint MyQ(List<Object> s, Query q, int operator) {
2.     if (s.size() == 2) {
3.         switch (operator) {
4.             case Constants.GREATER_OPERATOR:
5.                 return
   q.constrain(correctTypeConverter(s.get(0),
   s.get(1))).greater();
6.             case Constants.SMALLER_OPERATOR:
7.                 return
   q.constrain(correctTypeConverter(s.get(0),
   s.get(1))).smaller();
8.             case Constants.LIKE_OPERATOR:
9.                 return q.constrain(s.get(0)).like();
10.             case Constants.EQUALS_OPERATOR:
11.                 return
   q.constrain(correctTypeConverter(s.get(0), s.get(1)));
12.             case Constants.GREATER_EQUALS_OPERATOR:
13.                 return MyQ(s, q,
   Constants.GREATER_OPERATOR).or(MyQ(s, q,
   Constants.EQUALS_OPERATOR));
14.             case Constants.SMALLER_EQUALS_OPERATOR:
15.                 return MyQ(s, q,
   Constants.SMALLER_OPERATOR).or(MyQ(s, q,
   Constants.EQUALS_OPERATOR));
16.         }
17.     }
18.     Query sub = q.descend(s.get(0).toString());
19.     s.remove(0);
20.     return MyQ(s, sub, operator);
21. }
```

Εικόνα 86 "Μέθοδος MyQ"

Στην εσωτερική κλάση **RunQuery**, κώδικας της Εικόνας 87, θα κάνουμε χρήση των μεθόδων πριν την εκτέλεση, κατά την εκτέλεση και μετά την εκτέλεση, καθώς και ορισμένων βοηθητικών μεθόδων. Στην μέθοδο **"onPreExecute()"** αρχικοποιούμε τις μεταβλητές που θα χρησιμοποιήσουμε, τη λίστα τύπου ReflectFields **"reflectFields"**, τη λίστα τύπου αλφαριθμητικό **"userClasses"**, τη λίστα τύπου αλφαριθμητικό **"values"**, και στη συνέχεια αφού αρχικοποιήσουμε το αντικείμενο τύπου ProgressDialog για χρήση στην συγκεκριμένη Activity θέτουμε το μήνυμα που θα εμφανίζεται και τέλος το παρουσιάζουμε στο προσκήνιο.

```
1. class RunQuery extends AsyncTask<String, Void, Void> {
2.
3.     ProgressDialog mProgressDialog;
4.     List<ReflectField> reflectFields;
5.     List<String> values;
6.
7.     @Override
8.     protected void onPreExecute() {
9.         super.onPreExecute();
10.           reflectFields = new ArrayList<>();
11.           userClasses = new ArrayList<>();
12.           values = new ArrayList<>();
13.           mProgressDialog = new
   ProgressDialog(RecursivePrint.this);
14.           mProgressDialog.setIndeterminate(true);
15.           mProgressDialog.setTitle("Loading Results");
16.           mProgressDialog.setMessage("Your Results will be
   available as soon as possible");
17.           mProgressDialog.show();
18.       }
```

Εικόνα 87 "Κλάση RunQuery και Μέθοδος onPreExecute"

Στην μέθοδο **"doInBackground()"**, κώδικας της Εικόνας 88, εκτελείται η σύνδεση με τη Βάση, η λήψη των αποτελεσμάτων και ο τερματισμός της σύνδεσης. Αυτό συμβαίνει με τη χρήση της κλάσης Db4oSubClass, αρχικά δημιουργούμε ένα αντικείμενο τύπου **Db4oSubClass** περνώντας παραμετρικά τη μεταβλητή τύπου **Context** για τη διαχείριση των SharedPreferences από την Db4oSubClass. Μετατρέπουμε σε αντικείμενο τύπου λίστα αλφαριθμητικό τις τιμές της μεταβλητής **"classPath"**. Ελέγχουμε αν είναι κενή και καλούμε τη μέθοδο **"reflectFieldsNameASRF()"** περνώντας παραμετρικά τη μεταβλητή **"className"**, αφού όταν καλέσαμε την κλάση RunQuery περάσαμε την μεταβλητή **"className"** ως παράμετρο. Σε αντίθετη περίπτωση πέρνουμε το τελευταίο αντικείμενο της λίστας **"classPathList"** και το περνάμε παραμετρικά στη μέθοδο **"reflectFieldsNameASRF()"**. Στη συνέχεια αποθηκεύουμε στην μεταβλητή **"userClasses"** τύπου λίστας αλφαριθμητικό με τη βοήθεια της μεθόδου **"reflectClassesAsSTR()"** όλες τις κλάσεις που είναι αποθηκευμένες στη Βάση Δεδομένων και έχουν δημιουργηθεί από τον χρήστη. Έπειτα δημιουργούμε μια λίστα τύπου **ReflectClass** στην οποία με τη χρήση της μεθόδου **"reflectClass()"**, που δέχεται ως παράμετρο το όνομα της κλάσης ως αλφαριθμητικό αποθηκεύουμε ένα instance κάθε κλάσης που βρίσκεται στη μεταβλητή **"classPath"** και τερματίζουμε την σύνδεση με τον Εξυπηρετητή. Στη συνέχεια, κώδικας της Εικόνας 89, με τη βοήθεια της κλάσης **Db4oSubClass** εκκινούμε μία νέα σύνδεση με τον Εξυπηρετητή. Δημιουργούμε ένα αντικείμενο τύπου **Query** και θέτουμε ως κλάση εκκίνησης την κλάση που είχαμε επιλέξει στην κλάση Initial χωρίς να το εκτελέσουμε. Στη συνέχεια με τη χρήση του αντικειμένου τύπου

ConstraintsJsonData πέρνουμε τον operator που θα χρησιμοποιήσουμε, λογικό “και” ή λογικό “ή” και για κάθε περιορισμό που βρίσκεται στη λίστα περιορισμών του αντικειμένου τύπου **ConstraintsJsonData** θέτουμε στην λίστα **“s”** τύπου αντικειμένου (Object) τη διαδρομή (path), την τιμή που έχει θέσει ο χρήστης για τον περιορισμό και τον τύπο δεδομένων της μεταβλητής αυτής. Με τη μεταβλητή **“lastConstraint”** τύπου Constraint ορίζουμε τη μεταβλητή στην οποία θα προσθέτουμε κάθε περιορισμό που κατασκευάζουμε με τη χρήση της μεθόδου **“MyQ”**. Αν είναι ο πρώτος περιορισμός εκτελείται μόνο η μέθοδος “MyQ”, σε αντίθετη περίπτωση ανάλογα με τον operator που έχει θέσει ο χρήστης ενώνουμε το έως τώρα query, μεταβλητή **“lastConstraint”**, με το αποτέλεσμα της **“MyQ”**, που είναι το νέο Constraint, με τη χρήση λογικού “και” ή λογικού “ή” ανάλογα με την απόφαση του χρήστη. Τέλος, Σχήμα 90, εκτελούμε το query, με τη βοήθεια της **“CloseDB()”** τερματίζουμε τη σύνδεση με τον Εξυπηρετητή και αποθηκεύουμε το αποτέλεσμά του σε μία μεταβλητή τύπου **ObjectSet**, μία μορφή λίστας, ελέγχουμε αν είναι η πρώτη φορά που εκτελούμε το query, **refectClassIndex=-1**, και εμφανίζουμε όλα τα αντικείμενα που ταιριάζουν στους περιορισμούς μας με τη χρήση της μεθόδου **“printAllObjects()”**, που δέχεται ως παράμετρο ένα αντικείμενο τύπου **ObjectSet**, γραμμή εννέα. Σε αντίθετη περίπτωση, **reflectClassIndex διάφορο του -1**, τότε σε περίπτωση που δεν έχουμε ανοίξει κάποια κλάση πέραν της αρχικής θα εκτελεστεί η γραμμή έξι, περνώντας ως παράμετρο στη μέθοδο **“printObject()”** το επιλεγμένο αντικείμενο από τη λίστα όλων των αποτελεσμάτων. Στην περίπτωση που θελήσουμε να προβάλλουμε ένα αντικείμενο που αποτελεί αναφορά σε άλλο αντικείμενο της Βάσης Δεδομένων τότε θα εκτελέσουμε τη γραμμή τέσσερα, περνώντας ως παράμετρο το αποτέλεσμα της αναδρομικής μεθόδου **“findOutWhichObjectToPrint()”**, η οποία δέχεται ως είσοδο το αντικείμενο που επέλεξε ο χρήστης από το αρχικό query, μία λίστα τύπου αλφαριθμητικό που περιέχει τα ονόματα των πεδίων που έχουμε επιλέξει και μία λίστα τύπου **ReflectClass** που περιέχει τις κλάσεις στις οποίες ο χρήστης έθεσε περιορισμούς.

```
1. protected Void doInBackground(String... params) {
2.     Db4oSubClass db4oSubClass = new Db4oSubClass(ctx);
3.     List<String> classPathList = new
   ArrayList<>(Arrays.asList(classPath.split(":")));
4.     if (!classPathList.isEmpty()) {
5.         reflectFields =
   db4oSubClass.reflectFieldsNameASRF(classPathList.get(classPat
   hList.size() - 1));
6.     } else {
7.         reflectFields =
   db4oSubClass.reflectFieldsNameASRF(params[0]);
8.     }
9.     userClasses = db4oSubClass.reflectClassesAsSTR();
10.      List<ReflectClass> classPathReflectClasses = new
   ArrayList<>();
11.      if (reflectClassIndex != -1 && attributePath != null) {
12.          for (String className : new
   ArrayList<>(Arrays.asList(classPath.split(":")))) {
   classPathReflectClasses.add(db4oSubClass.reflectClass(classNa
   me));
13.          }
14.      }
15.      db4oSubClass.CloseDB();
```

Εικόνα 88 "Μέθοδος doInBackground (Α' Μέρος)"

```
1. db4oSubClass = new Db4oSubClass(ctx);
2. Query query = db4oSubClass.getDb().query();
3. query.constrain(db4oSubClass.reflectClass(params[0]));
4. if (constraintsJsonData != null) {
5.    int queryOperator = constraintsJsonData.getOperator();
6.    Constraint lasConstraint = null;
7.    for (MyConstraint myConstraint :
   constraintsJsonData.getConstraints()) {
8.        List<Object> s = new ArrayList<>();
9.        s.addAll(myConstraint.getPath());
10.          s.add(myConstraint.getValue());
11.          s.add(myConstraint.getReflectFieldType());
12.          s.remove(0);
13.          if ((lasConstraint != null)||(queryOperator==-1)) {
14.              switch (queryOperator) {
15.                  case Constants.AND_OPERATOR:
16.                      lasConstraint = lasConstraint.and(MyQ(s,
   query, myConstraint.getOperator()));
17.                      break;
18.                  case Constants.OR_OPERATOR:
19.                      lasConstraint = lasConstraint.or(MyQ(s,
   query, myConstraint.getOperator()));
20.                      break;
21.              }
22.          } else {
23.              lasConstraint = MyQ(s, query,
   myConstraint.getOperator());
24.          }
25.      }
26.  }
```

Εικόνα 89 "Μέθοδος doInBackground (Β' Μέρος)"

```
1. ObjectSet objectSet = query.execute();
2. if (reflectClassIndex != -1) {
3.    if (attributePath != null) {
4. printObject(findOutWhichObjectToPrint(objectSet.get(reflectCl
   assIndex), new
   ArrayList<>(Arrays.asList(attributePath.split(":"))),
   classPathReflectClasses));
5.    } else {
6.        printObject(objectSet.get(reflectClassIndex));
7.    }
8. } else {
9.    printAllObjects(objectSet);
10.   }
11.   db4oSubClass.CloseDB();
12.   return null;
```

Εικόνα 90 "Μέθοδος doInBackground (Γ' Μέρος)"

Οι μέθοδοι **“private void printAllObjects(ObjectSet objectSet)”** και **“private void printObject(Object o)”**, κώδικας της Εικόνας 91, είναι οι μέθοδοι εκτύπωσης. Η μέθοδος **“printAllObjects()”** δέχεται ως είσοδο ένα αντικείμενο τύπου **OjectSet** και εμφανίζει όλα τα αντικείμενα, δηλαδή το όνομα της κλάσης του εκάστοτε πεδίου. Είναι υπεύθυνη για την εκτύπωση των αποτελεσμάτων του πρώτου query που εκτελείται και περιέχει αντικείμενα της κλάσης που ο χρήστης χρησιμοποίησε ως κλάση ρίζα. Η μέθοδος **“printObject()”**, δέχεται ως είσοδο ένα αντικείμενο τύπου αντικειμένου (Object) και για κάθε πεδίο (ReflectField) που είναι αποθηκευμένο στη λίστα **“reflectfields”** τύπου ReflectField πέρνει την τιμή του πεδίου αυτού. Σε περίπτωση που αυτή είναι κενή (null) τυπώνεται σχετικό μήνυμα, διαφορετικά τυπώνεται η τιμή της μεταβλητής αυτής.

```
1. private void printAllObjects(ObjectSet objectSet) {
2.     for (Object o : objectSet) {
3.         values.add(o.toString());
4.     }
5. }
6.
7. private void printObject(Object o) {
8.     for (ReflectField reflectField : reflectFields) {
9.         Object value = reflectField.get(o);
10.          if (value != null) {
11.              values.add(value.toString());
12.          } else {
13.              values.add("null");
14.          }
15.      }
16.  }
```

Εικόνα 91 “Μέθοδοι printAllObjects και printObject”

Η μέθοδος **“private Object findOutWhichObjectToPrint(Object o, List<String> attributePath, List<ReflectClass> classPathReflectClasses)”**, κώδικας της Εικόνας 92, είναι μια αναδρομική μέθοδος που δέχεται ως είσοδο ένα αντικείμενο τύπου αντικειμένου (Object), μία λίστα τύπου αλφαριθμητικό και μία λίστα τύπου **ReflectClass**. Αποθηκεύει σε προσωρινή μεταβλητή τύπου **ReflectField** το πεδίο της πρώτης κλάσης που βρίσκεται στη λίστα **“classPathReflectClasses”** και έχει ως όνομα το πρώτο αντικείμενο της λίστας τύπου αλφαριθμητικό **“attributePath”**. Στην περίπτωση που είναι η τελευταία μεταβλητή που έχει η λίστα **“attributePath”**, γραμμή τρία τότε επιστρέφει την τιμή αυτού του πεδίου. Σε αντίθετη περίπτωση μειώνει κατά ένα αντικείμενο τόσο τι λίστα αντικειμένου τύπου αλφαριθμητικό, **“attributePath”**, όσο και της λίστας τύπου **ReflectClass**, **“classPathReflectClasses”** και τέλος καλούμε αναδρομικά την μέθοδο **“findOutWhichObjectToPrint()”** περνώντας ως παραμέτρους την τιμή του πεδίου

“reflectField”, την ανανεωμένη λίστα **“attributePath”** και την ανανεωμένη λίστα **“classPathReflectClasses”**.

```
1. private Object findOutWhichObjectToPrint(Object o,
   List<String> attributePath, List<ReflectClass>
   classPathReflectClasses) {
2.    ReflectField reflectField =
   classPathReflectClasses.get(0).getDeclaredField(attributePath
   .get(0));
3.    if (attributePath.size() == 1) {
4.        return reflectField.get(o);
5.    }
6.    attributePath.remove(0);
7.    classPathReflectClasses.remove(0);
8.    return findOutWhichObjectToPrint(reflectField.get(o),
   attributePath, classPathReflectClasses);
9. }
```

Εικόνα 92 “Μέθοδος findOutWhichObjectToPrint”

Στη συνέχεια η μέθοδος **“onPostExecute()”**, κομμάτια κώδικα των Εικόνων 93 και 94, θέτει σαν τίτλο του Activity είτε την τιμή **“RecursivePrint”** είτε την τιμή ***“ΟΝΟΜΑ_ΚΛΑΣΗΣ(ΑΡΙΘΜΟΣ_ΑΠΟΤΕΛΕΣΜΑΤΩΝ)”***, ανάλογα με την τιμή της μεταβλητής **“QueryFlag”**. Στην περίπτωση που το query δεν εκτελείται για πρώτη φορά, **reflectClassIndex διάφορο του -1**, τότε ορίζουμε στον **Adapter** τύπου ReflectFieldsValuesRecyclerViewAdapter τι θα συμβεί αν ο χρήστης επιλέξει κάποιο αντικείμενο από τη λίστα. Στην περίπτωση που αυτό έχει την τιμή **Null**, κενό αντικείμενο, εμφανίζεται σχετικό μήνυμα. Στην περίπτωση που ο χρήστης έχει ήδη ανοίξει το συγκεκριμένο αντικείμενο, δηλαδή συμβαίνει αυτό που φαίνεται στην Εικόνα 95 τότε εμφανίζουμε μήνυμα που ενημερώνει τον χρήστη ότι δεν μπορεί να ανοίξει το συγκεκριμένο αντικείμενο καθώς θα ξεκινήσει ένα ατέρμονα βρόγχο. Στην περίπτωση που δεν ισχύει κάτι από τα παραπάνω τότε εκκινούμε ένα νέο Activity της κλάσης **RecursivePrint** όπου ο χρήστης θα ανοίξει το αντικείμενο αναφοράς που επέλεξε, με παραμέτρους το όνομα της κλάσης ως αλφαριθμητικό, τη θέση του αντικειμένου που επέλεξε στη λίστα, τους περιορισμούς μαζί με τον operator ως αντικείμενα τύπου **ConstraintsJsonData** αφού πρώτα τα έχει μετασχηματίσει σε JSON αρχείο με τη χρήση του αντικειμένου **ObjectMapper**, τον τύπο δεδομένων, ουσιαστικά είναι το όνομα της κλάσης στην οποία αποτελεί αναφορά το πεδίο που επέλεξε, και τέλος τη διαδρομή (path) των πεδίων που έχει επιλέξει ο χρήστης. Στην περίπτωση που είναι η πρώτη φορά που εκτελείται το query, **reflectClassindex=-1**, κώδικας της Εικόνας 96, τότε ορίζουμε στον **Adapter** τύπου ReflectClassesResultsRecyclerViewAdapter τι θα συμβεί αν ο χρήστης επιλέξει κάποιο αντικείμενο από τη λίστα. Με την επιλογή κάποιου αντικειμένου από τη λίστα θα εκκινηθεί ένα νέο Activity της κλάσης **RecursivePrint**

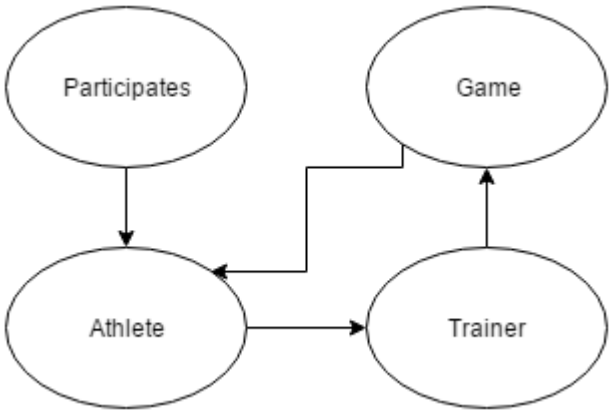
όπου ο χρήστης θα ανοίξει το αντικείμενο που επέλεξε με παραμέτρους το όνομα της κλάσης ως αλφαριθμητικό, τη θέση του αντικειμένου που επέλεξε στη λίστα και τους περιορισμούς μαζί με τον operator ως αντικείμενα τύπου **ConstraintsJsonData** αφού πρώτα τα έχει μετασχηματίσει σε JSON αρχείο με τη χρήση του αντικειμένου **ObjectMapper**. Τέλος, τερματίζουμε και επαναφέρουμε στο παρασκήνιο το αντικείμενο τύπου ProgressDialog.

```
1. protected void onPostExecute(Void tmpt) {
2.     super.onPostExecute(tmpt);
3.     if (QueryFlag) {
4.         setTitle(getTitle() + " ( " + values.size() + " )");
5.     } else {
6.         setTitle("RecursivePrint");
7.     }
8.     if (reflectClassIndex != -1) {
9.         reflectFieldsValuesRecyclerViewAdapter = new
   ReflectFieldsValuesRecyclerViewAdapter(values, reflectFields,
   new OnReflectFieldItemClickedListener() {
10.             public void onListItemClicked(String value,
   ReflectField reflectField) {
11.                 String fieldType =
   reflectField.getFieldType().getName();
12.                 if (value.equals("null")) {
13.                     Toast.makeText(RecursivePrint.this, "You
   cannot see a null object reference.",
   Toast.LENGTH_LONG).show();
14.                 } else if (userClasses.contains(fieldType)) {
15.                     List<String> tempL = new
   ArrayList<>(Arrays.asList(classPath.split(":")));
16.                     if (tempL.contains(fieldType)) {
17.                         AlertDialog.Builder
   alertDialogBuilder = new
   AlertDialog.Builder(getApplicationContext());
18.                         alertDialogBuilder.setTitle("You
   already have viewed " + fieldType);
19.                         alertDialogBuilder.setMessage("You
   cannot view " + fieldType + " again");
20.
   alertDialogBuilder.setNeutralButton("Close", new
   DialogInterface.OnClickListener() {
21.                             public void
   onClick(DialogInterface     dialog, int which) {
22.                                 dialog.dismiss();
23.                             }
24.                         });
25.                         AlertDialog alertDialog =
   alertDialogBuilder.create();
26.                         alertDialog.show();
27.                     }
```

Εικόνα 93 "Μέθοδος onPostExecute (Α' Μέρος)"

```
1.      else {
2.          try {
3.              Intent intent = new Intent(RecursivePrint.this,
   RecursivePrint.class);
4.              intent.putExtra("className", className);
5.             intent.putExtra("reflectClassIndex",
   reflectClassIndex);
6.             intent.putExtra("ConstraintsJsonData",
   mapper.writeValueAsString(constraintsJsonData));
7.            intent.putExtra("classPath", classPath + ":" +
   fieldType);
8.           if (attributePath == null) {
9.               intent.putExtra("attributePath",
   reflectField.getName());
10.            } else {
11.                intent.putExtra("attributePath", attributePath +
   ":" + reflectField.getName());
12.                    }
13.            startActivity(intent);
14.            } catch (JsonProcessingException e) {
15.                            e.printStackTrace();
16.                        }
17.                    }
18.                }
19.            }
20.        });
21.
   recursiveRecyclerView.setAdapter(reflectFieldsValuesRecyclerVi
   ewAdapter);
22.
   reflectFieldsValuesRecyclerViewAdapter.notifyDataSetChanged();
23.    }
```

Εικόνα 94 "Μέθοδος onPostExecute (Β' Μέρος)"



Εικόνα 95 "Ατέρμονος Βρόγχος στην Εμφάνιση των Αποτελεσμάτων"

```
1. else {
2.             ReflectClassesResultsRecyclerViewAdapter
   reflectClassesResultsRecyclerViewAdapter = new
   ReflectClassesResultsRecyclerViewAdapter(values, new
   OnReflectClassItemClickedListener() {
3.                 public void onListItemClicked(int
   reflectClassIndex) {
4.                     Intent intent = new
   Intent(RecursivePrint.this, RecursivePrint.class);
5.                     try {
6.                         intent.putExtra("className",
   className);
7.                         intent.putExtra("reflectClassIndex",
   reflectClassIndex);
8.                         intent.putExtra("ConstraintsJsonData",
   mapper.writeValueAsString(constraintsJsonData));
9.                     } catch (JsonProcessingException e) {
10.                         e.printStackTrace();
11.                     }
12.                     startActivity(intent);
13.                 }
14.             });
   recursiveRecyclerView.setAdapter(reflectClassesResultsRecycler
   ViewAdapter);
   reflectClassesResultsRecyclerViewAdapter.notifyDataSetChanged(
   );
15.         }
16.         mProgressDialog.dismiss();
17.     }
18. }
```

Εικόνα 96 "Μέθοδος onPostExecute (Γ' Μέρος)"

3.2.7 WatchMyConstraints.java

Η κλάση **"WatchMyConstraints"** αφορά την προβολή των περιορισμών που έχει θέσει ο χρήστης, κώδικας της Εικόνας 97, στις γραμμές δύο έως τέσσερα ορίζουμε τις global μεταβλητές που θα χρησιμοποιήσουμε. Με τη σειρά που εμφανίζονται αυτές είναι η μεταβλητή **"constraintsJsonData"** τύπου ConstraintsJsonData όπου θα αποθηκεύσουμε τους περιορισμούς καθώς και τον operator (λογικό "και" ή λογικό "ή") που ο χρήστης έθεσε στην κλάση ConstraintsActivity. Η μεταβλητή **mapper** τύπου ObjectMapper θα χρησιμοποιηθεί για την μετατροπή αντικειμένων Java σε αντικείμενα JSON και αντίστροφα. Τέλος, η μεταβλητή τύπου **Menu** θα χρησιμοποιηθεί για να διαχειριστούμε τα αντικείμενα που θα φιλοξενεί ο Drawer.

```
1. public class WatchMyConstraints extends AppCompatActivity
   implements NavigationView.OnNavigationItemSelectedListener {
2.
3.     private ConstraintsJsonData constraintsJsonData;
4.     private static ObjectMapper mapper = new ObjectMapper();
5.     private Menu menu;
```

Εικόνα 97 "Κλάση WatchMyConstraints"

Στη μέθοδο **"protected void onCreate(Bundle savedInstanceState)"**, κώδικας της Εικόνας 98, που εκτός των άλλων ορίζει και το Γραφικό Περιβάλλον που θα χρησιμοποιηθεί, γραμμή τρία. Στις γραμμές τέσσερα και πέντε αφού συνδέσουμε την επιθυμητή γραμμή εργαλείων με την αντίστοιχη μεταβλητή του προγράμματός μας την ορίζουμε ως την ενεργή γραμμή εργαλείων της Activity μας. Στις γραμμές οχτώ έως δεκατέσσερα ανακτούμε την τιμή του πεδίου **ConstraintsJsonData**. Στη συνέχεια ελέγχουμε ότι η μεταβλητή **"jsonData"** περιέχει κάποια κάποια τιμή και με τη χρήση του αντικειμένου **ObjectMapper** αποσειριοποιούμε τα δεδομένα και μεταφράζουμε το JSON αρχείο σε αντικείμενο τύπου ConstraintsJsonData. Στη γραμμή δεκαέξι καλούμε τη μέθοδο **"fillList()"** που θα γεμίσει τη λίστα του Drawer. Στο τέλος της μεθόδου **"onCreate()"** ορίζουμε το αντικείμενο **ActionBar** που θα χρησιμοποιηθεί από την Activity.

```
1. protected void onCreate(Bundle savedInstanceState) {
2.     super.onCreate(savedInstanceState);
3.     setContentView(R.layout.activity_watch_my_constraints);
4.     Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
5.     setSupportActionBar(toolbar);
6.     NavigationView navigationView = (NavigationView)
   findViewById(R.id.nav_viewQ);
7.     menu = navigationView.getMenu();
8.     String jsonData =
   getIntent().getStringExtra("ConstraintsJsonData");
9.     if (jsonData != null) {
10.          try {
11.              constraintsJsonData = mapper.readValue(jsonData,
   ConstraintsJsonData.class);
12.          } catch (IOException e) {
13.              e.printStackTrace();
14.          }
15.      }
16.      fillList();
17.      DrawerLayout drawer = (DrawerLayout)
   findViewById(R.id.drawer_layout);
18.      ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
19.              this, drawer, toolbar,
   R.string.navigation_drawer_open,
   R.string.navigation_drawer_close);
20.      drawer.setDrawerListener(toggle);
21.      toggle.syncState();
22.      navigationView.setNavigationItemSelectedListener(this); }
```

Εικόνα 98 "Μέθοδος onCreate"

Η μέθοδος **"private void fillList()"**, κώδικας της Εικόνας 99, έχει ως στόχο να δημιουργήσει τα αντικείμενα που θα εμφανίζονται στον Drawer. Αρχικά καθαρίζουμε τον Drawer απο τις προκαθορισμένες τιμές έτσι ώστε να υπάρχουν μόνο οι περιορισμοί. Η μεταβλητή **"counter"** τύπου ακέραιος αριθμός χρησιμοποιείται για να θέσουμε ένα μοναδικό αναγνωριστικό σε κάθε αντικείμενο του Drawer καθώς και για την ονοματοδοσία των αντικειμένων που εμφανίζονται σε αυτόν. Στις γραμμές τέσσερα έως επτά εκτελούμε μία επαναληπτική διαδικασία (for loop) όπου ορίζουμε για κάθε αντικείμενο του Drawer την τιμή και το ID του.

```
1. private void fillList(){
2.     menu.clear();
3.     int counter=1;
4.     for (MyConstraint myConstraint :
   constraintsJsonData.getConstraints()) {
5.
   menu.add(Menu.NONE,counter,Menu.NONE,getString(R.string.Const
   raint)+counter);
6.         counter++;
7.     }
8.     onNavigationItemSelected(menu.getItem(0));
9. }
```

Εικόνα 99 "Μέθοδος fillList"

Η μέθοδος **"public boolean onNavigationItemSelected(MenuItem item)"**, κώδικας της Εικόνας 100, εκτελείται κάθε φορά που ο χρήστης επιλέγει από τον Drawer ένα αντικείμενο. Η μεταβλητή τύπου **Menu** που δέχεται ως είσοδο έχει αποθηκευμένες όλες τις πληροφορίες για το επιλεγμένο αντικείμενο. Στη γραμμή δύο ανακτούμε το ID που ορίσαμε στη μέθοδο **"fillList()"**. Στη γραμμή τέσσερα κλείνουμε το παράθυρο του Drawer και στη γραμμή πέντε συνδέουμε το αντικείμενο τύπου WebView που ορίσαμε για το Γραφικό Περιβάλλον με το αντικείμενο τύπου **WebView** που θα χρησιμοποιήσουμε για να παραμετροποιήσουμε το πρώτο. Στη γραμμή έξι δηλώνουμε ότι θέλουμε να φορτώσουμε στο αντικείμενο WebView ένα αντικείμενο τύπου αλφαριθμητικό, το αποτέλεσμα της μεθόδου **"fillData"** που δέχεται ως είσοδο τη θέση του επιλεγμένου αντικειμένου στη λίστα των περιορισμών, δηλώνουμε επίσης ότι θα εμφανίζεται ως απλό κείμενο με μορφοποίηση HTML, η τελευταία μεταβλητή αφορά την κωδικοποίηση και καθώς δεν χρησιμοποιούμε κάποιο εξειδικευμένο χαρακτήρα το αφήνουμε κενό, τιμή **Null**. Στις γραμμές επτά έως οχτώ δηλώνουμε ότι επιθυμούμε το αντικείμενο τύπου WebView θέλουμε να έχει λειτουργικότητα μεγέθυνσης. Τέλος, στη γραμμή εννέα επαναφορτώνουμε το περιεχόμενο του αντικειμένου τύπου WebView.

```
1. public boolean onNavigationItemSelected(MenuItem item) {
2.     int id = item.getItemId()-1;
3.     DrawerLayout drawer = (DrawerLayout)
   findViewById(R.id.drawer_layout);
4.     drawer.closeDrawer(GravityCompat.START);
5.     WebView
   tmpView=(WebView)findViewById(R.id.constraintsViewer);
6.     tmpView.loadData(fillData(id), "text/html", null);
7.     tmpView.invokeZoomPicker();
8.     tmpView.getSettings().setBuiltInZoomControls(true);
9.     tmpView.reload();
10.       return true;
11.   }
```

Εικόνα 100 "Μέθοδος onNavigationItemSelected"

Η μέθοδος **"private String fillData(int position)"**, κώδικας της Εικόνας 101, λαμβάνει ως είσοδο τη θέση του επιλεγμένου αντικειμένου από τον Drawer και στη συνέχεια δημιουργεί τον HTML κώδικα που θα φορτωθεί στο αντικείμενο τύπου WebView. Στην δεύτερη γραμμή αποθηκεύουμε το επιλεγμένο αντικείμενο τύπου **MyConstraint** από τη λίστα των περιορισμών. Η μεταβλητή **"css_content"** είναι τύπου αλφαριθμητικό και περιέχει τον κώδικα που θα μορφοποιήσει τα απλά στοιχεία. Έτσι μπορούμε να αλλάξουμε τη μορφοποίηση αλλάζοντας απλά το κείμενο της μεταβλητής "css_content". Στη γραμμή τέσσερα αποθηκεύουμε τον αριθμό των περιορισμών που έχει ορίσει ο χρήστης, για χρήση αυτού στα Loop που ακολουθούν. Στη γραμμή πέντε αρχικοποιούμε την μεταβλητή **"tmp"** τύπου αλφαριθμητικό, την οποία θα χρησιμοποιήσουμε για να κτίσουμε τον κώδικα HTML. Στις γραμμές έξι έως οχτώ χρησιμοποιούμε ένα Loop το οποίο δημιουργεί αντικείμενα τύπου λίστα μέχρι και πριν το πεδίο στο οποίο ο χρήστης έχει θέσει περιορισμό. Στη γραμμή εννέα τυπώνουμε το όνομα της μεταβλητής στην οποία ο χρήστης έχει θέσει τον περιορισμό, τον operator που έχει επιλέξει ο χρήστης καθώς και την τιμή περιορισμού. Στη συνέχεια γραμμές δέκα έως δώδεκα με τη χρήση ενός Loop κλείνουμε όλα τα αντικείμενα τύπου λίστα που δημιουργήσαμε. Τέλος, κλείνουμε τα βασικά στοιχεία της HTML δομής και επιστρέφουμε το αποτέλεσμα.

```
1. private String fillData(int position){
2.    MyConstraint
   tmpCon=constraintsJsonData.getConstraints().get(position);
3.    String css_content="/*Now the CSS*/";
4.    String css="<style>"+css_content+"</style>";
5.    int size=tmpCon.getPath().size();
6.    String tmp="<html>"+css+"<body><div class=\"tree\">";
7.    for (int i=0;i<size-1;i++) {
8.        tmp += "<ul><li><a
   href=\"#\">"+tmpCon.getPath().get(i)+"</a>";
9.    }
10.      tmp+="<ul><li><a
   href=\"#\">"+tmpCon.getPath().get(size-1)+addOperator(tmpCon.
   getOperator())+tmpCon.getValue()+"</a>";
11.      for(int i1=0; i1<size;i1++){
12.          tmp+="</li></ul>";
13.      }
14.      tmp+="</div></body></html>";
15.      return tmp;
16.  }
```

Εικόνα 101 "Μέθοδος fillData"

Η μέθοδος **"private String addOperator(int operator)"**, κώδικας της Εικόνας 102, δέχεται ως είσοδο τον κωδικό του operator έτσι όπως έχει δηλωθεί στην κλάση **Constants** και επιστρέφει ως αλφαριθμητικό το αντίστοιχο σύμβολο. Αυτό επιτυγχάνεται με τη χρήση ενός **switch - case** statement.

```
1. private String addOperator(int operator){
2.     String tmpOperator=null;
3.     switch (operator) {
4.         case Constants.GREATER_OPERATOR:
5.             tmpOperator=" > ";
6.             break;
7.         case Constants.SMALLER_OPERATOR:
8.             tmpOperator=" < ";
9.             break;
10.          case Constants.LIKE_OPERATOR:
11.              tmpOperator=" LIKE ";
12.              break;
13.          case Constants.EQUALS_OPERATOR:
14.              tmpOperator=" = ";
15.              break;
16.          case Constants.GREATER_EQUALS_OPERATOR:
17.              tmpOperator=" >= ";
18.              break;
19.          case Constants.SMALLER_EQUALS_OPERATOR:
20.              tmpOperator=" <= ";
21.              break;
22.      }
23.      return tmpOperator;
24.  }
```

Εικόνα 102 "Μέθοδος addOperator"

3.3 Βασικά αρχεία που αφορούν το Γραφικό Περιβάλλον της Εφαρμογής

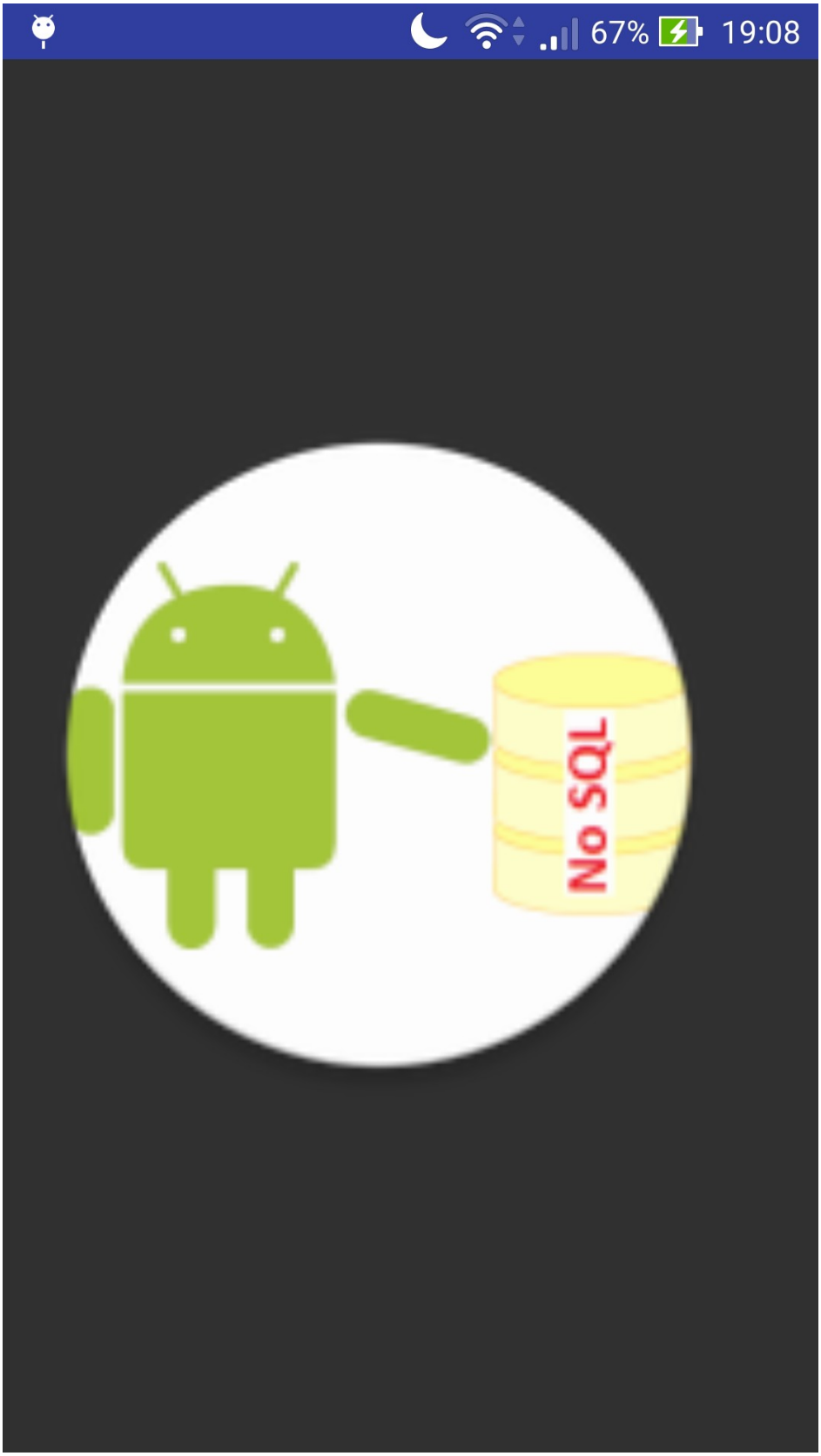
3.3.1 Activity Splash Layout

Στο αρχείο "activity_splash.xml" είναι αποθηκευμένα τα Views που χρειαζόμαστε για να αλληλεπιδράσει ο κώδικας που είδαμε στο προηγούμενο υποκεφάλαιο με τον χρήστη κατά την έναρξη της εφαρμογής. Τα Views είναι τα βασικά δομικά στοιχεία για το Γραφικό Περιβάλλον χρήστη και αποτελούν μια υπερκλάση την οποία κληρονομούν όλα τα αντικείμενα που προβάλλονται στην οθόνη μίας συσκευής με το Λειτουργικό Σύστημα Android. Ένα View αποτελεί ένα ορθογώνιο σχήμα που καταλαμβάνει χώρο στην οθόνη και είναι υπεύθυνο για τη δημιουργία γραφικών στοιχείων καθώς και για την διαχείριση των γεγονότων που λαμβάνουν χώρα όταν ο χρήστης αλληλεπιδρά με αυτό. Στις γραμμές ένα έως έντεκα, κώδικας της Εικόνας 103, ορίζουμε τα βασικά χαρακτηριστικά του αρχείου οδηγιών για τη δημιουργία του Γραφικού Περιβάλλοντος. Αυτά είναι με τη σειρά το root Layout που θα χρησιμοποιηθεί και θα περικλύει όλα τα υπόλοιπα Views, στην

συγκεκριμένη περίπτωση επιλέξαμε να χρησιμοποιήσουμε το Layout Linearlayout που ομαδοποιεί όλα τα παιδιά του (children Views) σε μία σειρά. Στη συνέχεια η χρήση ενός XML namespace που θα χρησιμοποιηθεί ως κομμάτι αναφοράς για το Android SDK. Οι επόμενες δύο γραμμές ενημερώνουν το Λειτουργικό Σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει τόσο χώρο οριζόντια και κατακόρυφα όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“match_parent”** που σημαίνει ότι θα καταλάβει τον ίδιο χώρο που διαθέτει το αρχικό View της εφαρμογής. Η μεταβλητή **gravity** δηλώνει τη θέση που θα έχει το εκάστοτε View στο χώρο, στη συγκεκριμένη περίπτωση δηλώνουμε ότι όλα τα παιδία Views αυτού θα είναι τοποθετημένα στο κέντρο οριζοντίως. Στην επόμενη γραμμή ορίζουμε τη μεταβλητή **“orientation”** που ενημερώνει το Λειτουργικό Σύστημα Android σχετικά με τον τρόπο που θα προβάλλεται, στην συγκεκριμένη περίπτωση επιλέξαμε την τιμή **vertical** καθώς θέλουμε κάθε View να καταλαμβάνει μία νέα σειρά. Οι επόμενες τέσσερεις γραμμές αφορούν το **Padding** που θα έχει το κάθε View από αριστερά, δεξιά, κάτω και πάνω, για να δηλωθεί το Padding στις κατευθύνσεις αριστερά, δεξιά και κάτω θα πρέπει να έχει δηλωθεί πρώτα η κατεύθυνση πάνω. Τέλος, η μεταβλητή **context** αφορά αναφορά στη κλάση στην οποία αποτελεί αυτό το αρχείο το Γραφικό Περιβάλλον. Στο συγκεκριμένο Fragment υπάρχει μόνο ένα αντικείμενο τύπου **ImageView** που απεικονίζει την βασική εικόνα της εφαρμογής. Όπως φαίνεται στην Εικόνα 104, ο χρήστης βλέπει την βασική εικόνα της εφαρμογής έως ότου περάσει το χρονικό περιθώριο που έχουμε δηλώσει.

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:tools="http://schemas.android.com/tools"
4.     android:layout_width="match_parent"
5.     android:layout_height="match_parent"
6.     android:paddingBottom="@dimen/activity_vertical_margin"
7.     android:paddingLeft="@dimen/activity_horizontal_margin"
8.     android:paddingRight="@dimen/activity_horizontal_margin"
9.     android:paddingTop="@dimen/activity_vertical_margin"
10.
   tools:context="com.example.finalthesis.db4o_the_project.Splas
   h"
11.       android:weightSum="1">
12.
13.
14.       <ImageView
15.           android:layout_width="wrap_content"
16.           android:layout_height="352dp"
17.           android:id="@+id/imageView2"
18.           android:src="@mipmap/ic_launcher_2"
19.           android:layout_weight="0.90"
20.           android:layout_gravity="center" />
21.   </LinearLayout>
```

Εικόνα 103 "Κώδικας του αρχείου activity_splash.xml"



Εικόνα 104 "Γραφικό Περιβάλλον του αρχείου activity_splash.xml"

3.3.2 Activity Login Layout

Στο αρχείο “activity_login.xml” είναι αποθηκευμένα τα Views που χρειαζόμαστε για να αλληεπιδράσει ο κώδικας που είδαμε στο προηγούμενο υποκεφάλαιο με τον χρήστη για να υποβάλλει τα στοιχεία σύνδεσης με τη Βάση Δεδομένων. Στις γραμμές ένα έως έντεκα, κώδικας της Εικόνας 105, ορίζουμε τα βασικά χαρακτηριστικά του του Γραφικού Περιβάλλοντος. Αυτά είναι με τη σειρά το root Layout που θα χρησιμοποιηθεί και θα περικλύει όλα τα υπόλοιπα Views, στην συγκεκριμένη περίπτωση επιλέξαμε να χρησιμοποιήσουμε το Layout **Linearlayout**. Αφού δηλώσουμε τις βασικές παραμέτρους όπως είδαμε και στην προηγούμενη παράγραφο, για την ενημέρωση του χρήστη ανάλογα με την εξέλιξη της διαδικασίας χρησιμοποιούμε ένα View τύπου ProgressBar. Τα βασικά χαρακτηριστικά αυτού του View είναι το ID που χρησιμοποιείται για να το κάνουμε bind με την αντίστοιχη μεταβλητή που χρησιμοποιούμε στην Java κλάση για να προσδόσουμε λειτουργικότητα και αποθηκεύεται στην κλάση R. Το View ProgressBar διαθέτει τέσσερεις επιλογές για προβολή, μία οριζόντια μπάρα που χρωματίζεται ανάλογα με το ποσοστό εξέλιξης της διαδικασίας και ένα κυκλικό αντικείμενο που πληροφορεί το χρήστη ότι μία διαδικασία βρίσκεται σε εξέλιξη και αποτελείται από τρία μεγέθη, μικρό μεσαίο, μεγάλο και αφορά στο χώρο που θα καταλάβει το View στην οθόνη, εμείς επιλέξαμε τη χρήση του μεγάλου κυκλικού αντικειμένου. Οι επόμενες δύο γραμμές ενημερώνουν το Λειτουργικό Σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει τόσο χώρο οριζόντια και κατακόρυφα όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“wrap_content”** που σημαίνει ότι θα καταλάβει τόσο χώρο όσο χρειάζεται για την προβολή του View. Τέλος, επειδή θέλουμε αυτό το View να εμφανίζεται μόνο όταν λαμβάνει χώρα η διαδικασία για τη σύνδεση με τη Βάση Δεδομένων, θέτουμε ως προκαθορισμένο να μην είναι ορατό στο χρήστη παραμόνο όταν συμβαίνει το παραπάνω.

```
1. <LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:tools="http://schemas.android.com/tools"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:gravity="center_horizontal"
6.     android:orientation="vertical"
7.     android:paddingBottom="@dimen/activity_vertical_margin"
8.     android:paddingLeft="@dimen/activity_horizontal_margin"
9.     android:paddingRight="@dimen/activity_horizontal_margin"
10.       android:paddingTop="@dimen/activity_vertical_margin"
11.    tools:context="com.example.finalthesis.db4o_the_project.Logi
   nActivity">
12.       <ProgressBar
13.           android:id="@+id/login_progress"
14.           style="?android:attr/progressBarStyleLarge"
15.           android:layout_width="wrap_content"
16.           android:layout_height="wrap_content"
17.           android:layout_marginBottom="8dp"
18.           android:visibility="gone" />
19.       <ScrollView
20.           android:id="@+id/login_form"
21.           android:layout_width="match_parent"
22.           android:layout_height="match_parent">
23.           <LinearLayout
24.               android:id="@+id/email_login_form"
25.               android:layout_width="match_parent"
26.               android:layout_height="wrap_content"
27.               android:orientation="vertical">
```

Εικόνα 105 "Κώδικας του αρχείου activity_login.xml (Α' Μέρος)"

Επειδή θα χρειαστεί να προβάλουμε μια σειρά από Views και ανάλογα με το μέγεθος της οθόνης μπορεί να μην χωρέσουν σε αυτή, χρησιμοποιούμε ένα αντικείμενο **ScrollView** που μας δίνει τη δυνατότητα να κάνουμε scroll down ή scroll up και να προβάλλουμε πολλαπλά αντικείμενα. Οι επόμενες δύο γραμμές ενημερώνουν το Λειτουργικό Σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει τόσο χώρο οριζόντια και κατακόρυφα όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **"match_parent"**. Επειδή το View ScrollView μπορεί να έχει μόνο ένα παιδί περικλύουμε όλα τα Views που θέλουμε να τοποθετήσουμε εντός του ScrollView σε ένα **LinearLayout**. Οι επόμενες δύο γραμμές ενημερώνουν το Λειτουργικό Σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει οριζόντια και κατακόρυφα χώρο όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **"match_parent** και **"wrap_content"** αντίστοιχα. Τέλος, ορίζουμε η μεταβλητή **"orientation"** να λαμβάνει την τιμή **vertical** έτσι ώστε τα αντικείμενα παιδιά του να εμφανίζονται κάθετα.

Τα παιδιά του LinearLayout, κώδικας της Εικόνας 106, είναι μια σειρά από View τύπου AutoCompleteTextView που μοιράζονται κοινά χαρακτηριστικά και οι μόνες διαφορές εντοπίζονται στα πεδία ID και hint, τα οποία μεταβάλλονται ανάλογα με την λειτουργικότητα του εκάστοτε View. Τα Views αυτά χρησιμοποιούνται ως είσοδος για τις τιμές διεύθυνση Εξυπηρετητή, πόρτα στην οποία ο Εξυπηρετητής αναμένει ερωτήματα και όνομα χρήστη. Τα κοινά χαρακτηριστικά τους έγγυνται στο γεγονός ότι και τα τρία κάνουν χρήση της βιβλιοθήκης **“android.support.design.widget.TextInputLayout”**, με τη χρήση αυτής της βιβλιοθήκης μας δίνεται η δυνατότητα όταν ο χρήστης γράψει εντός του πεδίου αυτού η μεταβλητή hint να απεικονίζεται μεν αλλά να βρίσκεται απομακρυσμένη ως floating Label. Αφού ορίσουμε το ID ενημερώνουμε το Λειτουργικό Σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει οριζόντια και κατακόρυφα χώρο όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“match_parent”** και **“wrap_content”** αντίστοιχα. Η μεταβλητή **“hint”** αφορά το κείμενο που θέλουμε να προβάλλεται στο εκάστοτε View. Επειδή θέλουμε ο χρήστης να περιορίζεται στον ορισμό μόνο ενός αντικειμένου κάθε φορά έχουμε ορίσει ότι κάθε View τύπου AutoCompleteTextView θα έχει μόνο μια σειρά, σε περίπτωση που δεν είχαμε δηλώσει τις δύο παρακάτω γραμμές ο χρήστης θα μπορούσε να δώσει πλέον της μιας τιμής ως multiline κείμενο. Έπειτα, κώδικας της Εικόνας 107, χρησιμοποιούμε ένα View τύπου EditText για την εισαγωγή από τον χρήστη του κωδικού πρόσβασης στη Βάση Δεδομένων. Όπως και στα αντικείμενα AutoComplete έτσι και στο EditText χρησιμοποιούμε τη βιβλιοθήκη “android.support.design.widget.TextInputLayout”. Ο λόγος που χρησιμοποιήσαμε το View EditText έναντι του View AutoCompleteTextView είναι το γεγονός ότι το View AutoCompleteTextView δεν υποστηρίζει τη χρήση τελιών αντί του κειμένου που ουσιαστικά εισάγει ο χρήστης, αυτό το χρειαζόμαστε καθώς αποτελεί τακτική των δημιουργών Γραφικού Περιβάλλοντος να μην εμφανίζεται το κείμενο που αποτελεί τον κωδικό αλλά αντί αυτού κάποια σύμβολα. Οι επόμενες δύο γραμμές ενημερώνουν το Λειτουργικό Σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει οριζόντια και κατακόρυφα χώρο όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“match_parent”** και **“wrap_content”** αντίστοιχα. Επειδή θέλουμε την λειτουργικότητα που περιγράψαμε παραπάνω χρησιμοποιούμε την μεταβλητή **“android:inputType="textPassword"”**, όπου δηλώνουμε ότι θα εισάγουμε κείμενο τύπου κωδικός. Στο τέλος αυτού του View Linearlayout έχουμε τα δύο κουμπιά που αφορούν την σύνδεση με τη Βάση Δεδομένων και την προβολή του σύντομου οδηγού χρήσης της εφαρμογής αντίστοιχα.

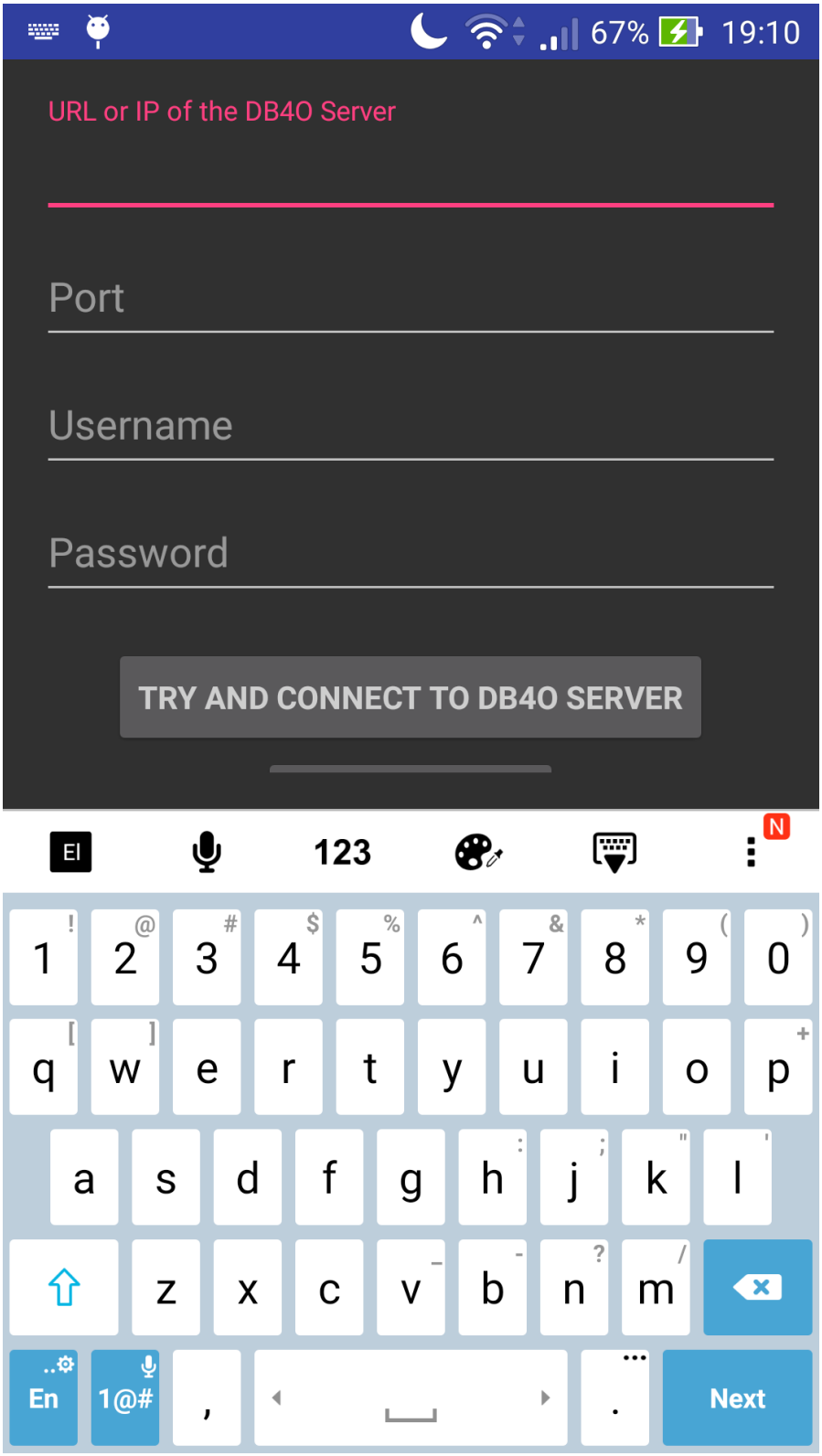
```
1. <android.support.design.widget.TextInputLayout
2.     android:layout_width="match_parent"
3.     android:layout_height="wrap_content">
4.     <AutoCompleteTextView
5.         android:id="@+id/LoginServerUser"
6.         android:layout_width="match_parent"
7.         android:layout_height="wrap_content"
8.         android:hint="@string/prompt_server"
9.         android:maxLines="1"
10.            android:singleLine="true" />
11.  </android.support.design.widget.TextInputLayout>
12.  <android.support.design.widget.TextInputLayout
13.     android:layout_width="match_parent"
14.     android:layout_height="wrap_content">
15.     <AutoCompleteTextView
16.         android:id="@+id/LoginPort"
17.         android:layout_width="match_parent"
18.         android:layout_height="wrap_content"
19.         android:hint="@string/prompt_port"
20.         android:maxLines="1"
21.         android:singleLine="true" />
22.  </android.support.design.widget.TextInputLayout>
23.  <android.support.design.widget.TextInputLayout
24.     android:layout_width="match_parent"
25.     android:layout_height="wrap_content">
26.     <AutoCompleteTextView
27.         android:id="@+id/LoginUser"
28.         android:layout_width="match_parent"
29.         android:layout_height="wrap_content"
30.         android:hint="@string/prompt_user"
31.         android:maxLines="1"
32.         android:singleLine="true" />
33.  </android.support.design.widget.TextInputLayout>
```

Εικόνα 106 “Κώδικας του αρχείου activity_login.xml (Β’ Μέρος)”

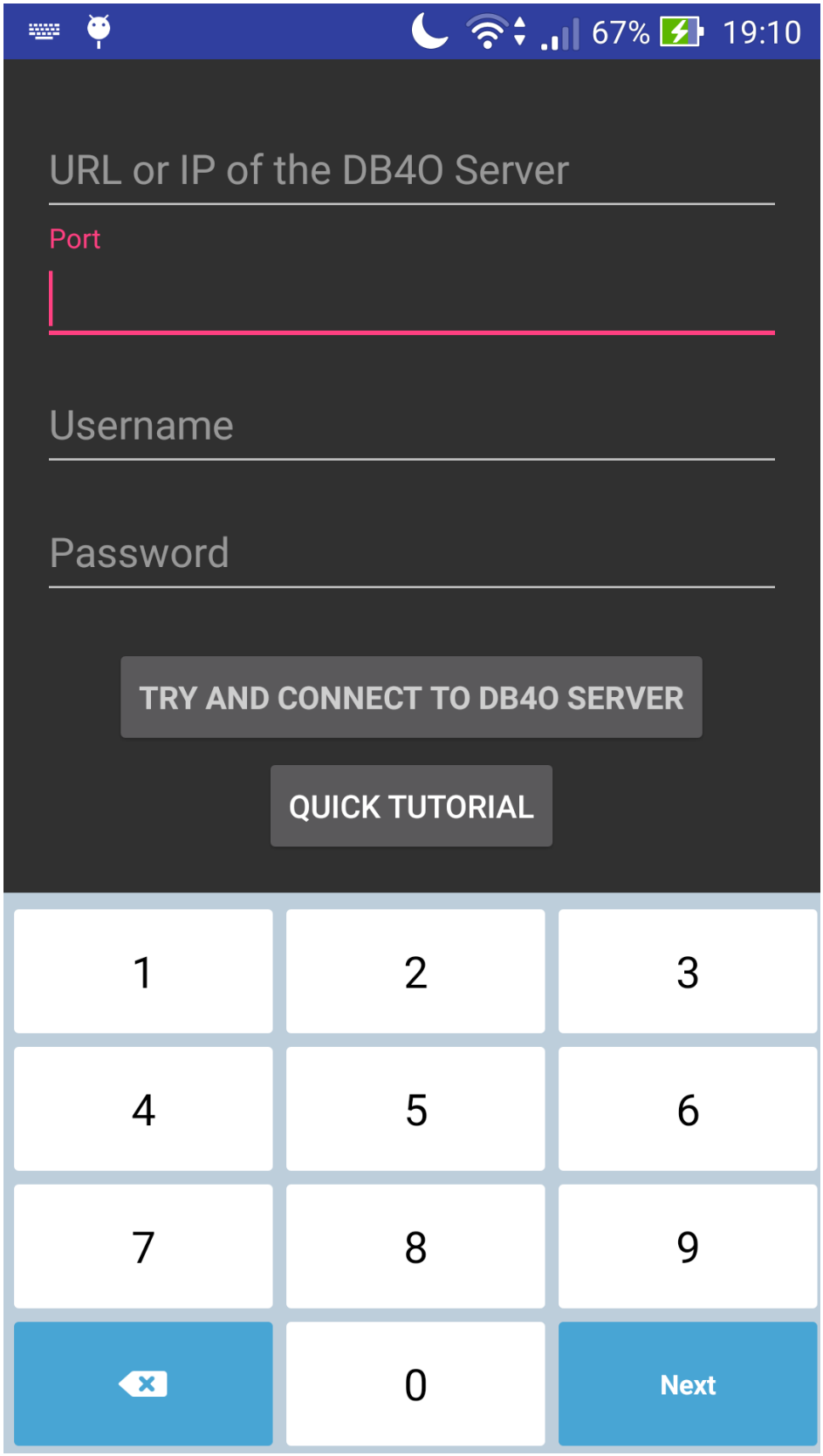
```
1. <android.support.design.widget.TextInputLayout
2.                 android:layout_width="match_parent"
3.                 android:layout_height="wrap_content">
4.                 <EditText
5.                     android:id="@+id/LoginPassword"
6.                     android:layout_width="match_parent"
7.                     android:layout_height="wrap_content"
8.                     android:hint="@string/prompt_password"
9.                     android:maxLines="1"
10.                       android:singleLine="true"
11.                       android:inputType="textPassword"/>
12.             </android.support.design.widget.TextInputLayout>
13.             <Button
14.                 android:id="@+id/email_sign_in_button"
15.                 style="?android:textAppearanceSmall"
16.                 android:layout_width="wrap_content"
17.                 android:layout_height="wrap_content"
18.                 android:layout_marginTop="16dp"
19.                 android:text="@string/action_sign_in"
20.                 android:textStyle="bold"
21.                 android:layout_gravity="center"
22.                 android:backgroundTint="@color/colorAccent"/>
23.             <Button
24.                 style="?android:attr/buttonStyleSmall"
25.                 android:layout_width="wrap_content"
26.                 android:layout_height="wrap_content"
27.                 android:text="@string/FAQ_TEXT"
28.                 android:id="@+id/FAQB"
29.                 android:enabled="true"
30.                 android:layout_gravity="center"
31.                 android:backgroundTint="@color/colorAccent"/>
32.         </LinearLayout>
33.     </ScrollView>
34. </LinearLayout>
```

Εικόνα 107 "Κώδικας του αρχείου activity_login.xml (Γ' Μέρος)"

Στο Γραφικό Περιβάλλον της Activity αυτής, Εικόνες 108 και 109, ο χρήστης εισάγει στα αντίστοιχα πεδία τις επιθυμητές τιμές για τα πεδία διεύθυνση Εξυπηρετητή, πόρτα στην οποία ο Εξυπηρετητής αναμένει ερωτήματα, όνομα χρήστη και κωδικό πρόσβασης. Αξίζει να σημειωθεί ότι ενώ στα πεδία διεύθυνση Εξυπηρετητή, όνομα χρήστη και κωδικό πρόσβασης το πληκτρολόγιο είναι τύπου αλφαριθμητικό καθώς ο χρήστης μπορεί να εισάγει συνδυασμό χαρακτήρων διαφορετικού είδους (γράμματα, αριθμούς, χαρακτήρες). Στην περίπτωση που ο χρήστης θέλει να εισάγει τιμή στο πεδίο πόρτα που αναμένει ο Εξυπηρετητής ερωτήματα, επειδή η μόνη αποδεκτή τιμή είναι αριθμός εμφανίζεται μόνο αριθμητικό πληκτρολόγιο. Όταν ο χρήστης πατήσει το κουμπί **"Try and connect"** θα γίνει επαλήθευση των στοιχείων που έδωσε και θα εμφανιστεί σχετικό μήνυμα επιτυχούς ή όχι σύνδεσης και αντίστοιχα θα αποθηκευτούν ή όχι τα στοιχεία που έδωσε. Στην περίπτωση που τα στοιχεία είναι λανθασμένα ο χρήστης θα παγιδευτεί σε αυτή την Activity μέχρι να δώσει τα ορθά, αντίθετα μεταβαίνει στην επόμενη Activity, την Initial.



Εικόνα 108 "Γραφικό Περιβάλλον του αρχείου activity_login.xml (Αλφαριθμητικό Πληκτρολόγιο)"



Εικόνα 109 "Γραφικό Περιβάλλον του αρχείου activity_login.xml (Αριθμητικό Πληκτρολόγιο)"

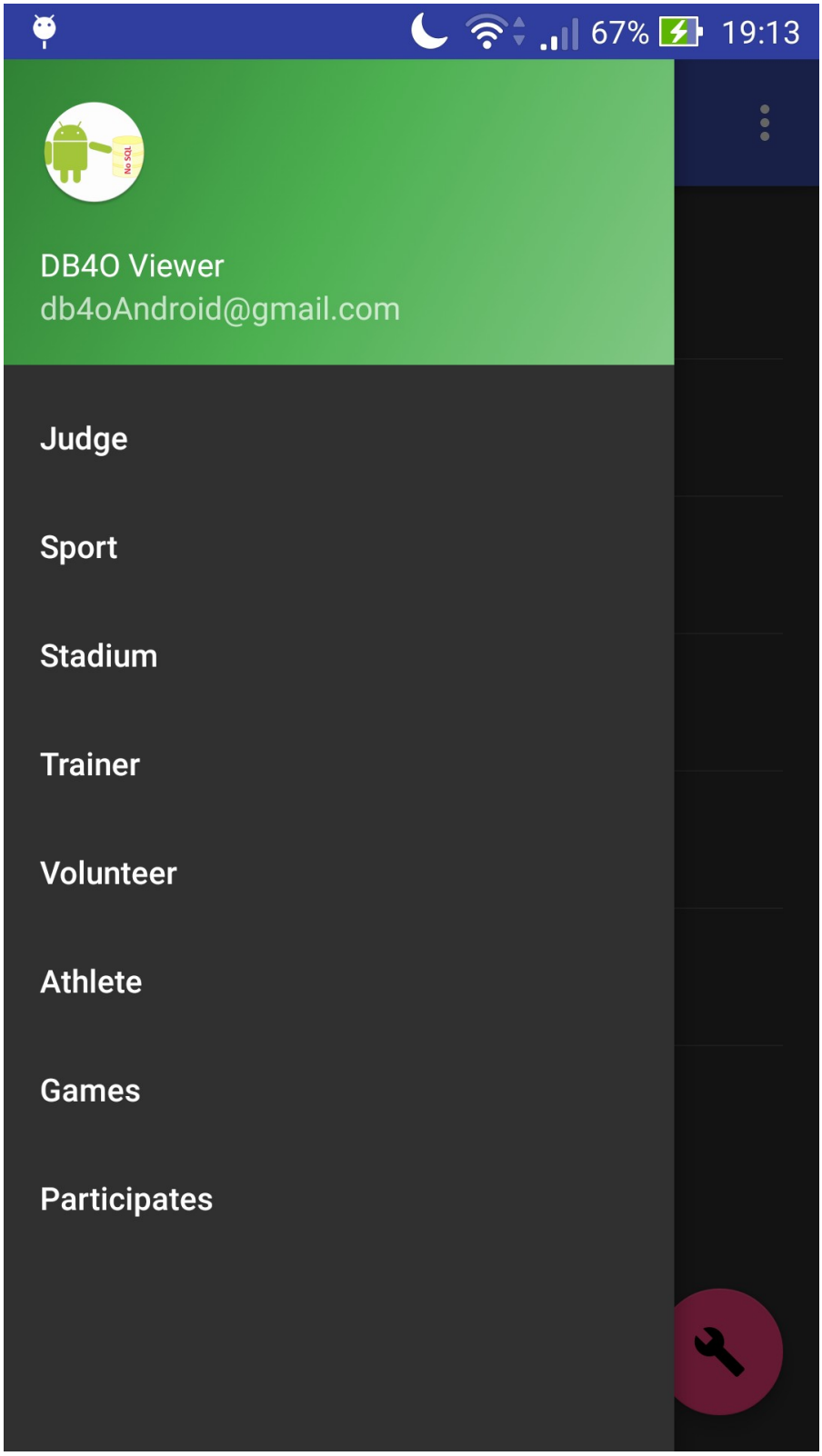
3.3.3 Activity Initial Layout

Στο αρχείο “activity_initial.xml” είναι αποθηκευμένα τα Views που χρειαζόμαστε για να αλληλεπιδράσει ο κώδικας που είδαμε στο προηγούμενο υποκεφάλαιο με τον χρήστη για να επιλέξει την κλάση που επιθυμεί από τη Βάση Δεδομένων, αντικείμενο που έχει κατασκευάσει ο ίδιος. Στις γραμμές ένα έως δέκα, κώδικας της Εικόνας 110, ορίζουμε τα βασικά χαρακτηριστικά του Γραφικού Περιβάλλοντος. Στην συγκεκριμένη περίπτωση επιλέξαμε να χρησιμοποιήσουμε το **“android.support.v4.widget.DrawerLayout”** που αποτελεί τη μόνη επιλογή σε περίπτωση χρήσης του NavigationView και των δυνατοτήτων που προσφέρει. Οι επόμενες δύο γραμμές ενημερώνουν το λειτουργικό σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει τόσο χώρο οριζόντια και κατακόρυφα όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“match_parent”**. Η μεταβλητή **“android:fitsSystemWindows”** σημαίνει ότι το background του drawer όταν τον ανοίγουμε θα είναι διαφανές. Με τη χρήση της μεταβλητής **“tools:openDrawer="start"”** δηλώνουμε στο Λειτουργικό Σύστημα ότι θέλουμε να έχουμε πρόσβαση στον Drawer, να είναι ενεργοποιημένος, στην περίπτωση που παραλείπαμε αυτή τη γραμμή ο χρήστης δεν θα μπορούσε να χρησιμοποιήσει τον Drawer. Στη συνέχεια το tag **“include”** δηλώνει ότι σε αυτό το σημείο μπορούμε να τοποθετήσουμε ένα Fragment ή Layout και ορίζει ότι θα πρέπει να καταλαμβάνει τόσο χώρο οριζόντια και κατακόρυφα όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“match_parent”**. Το tag “include” είναι ένα πολύ χρήσιμο εργαλείο το οποίο μας επιτρέπει να χτίσουμε διαφορετικά κομμάτια σε Fragment και στη συνέχεια να τα επαναχρησιμοποιήσουμε σε όσες Activity επιθυμούμε κάνοντας όσες συνθέσεις επιθυμούμε. Τέλος, το View **“android.support.design.widget.NavigationView”** είναι το View του Drawer όπου θα εμφανίζονται οι κλάσεις και θα βρίσκεται στην αριστερή πλευρά και θα εμφανίζεται καθ’ επιλογή του χρήστη. Οι επόμενες δύο γραμμές ενημερώνουν το λειτουργικό σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει οριζόντια και κατακόρυφα χώρο όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“wrap_content”** και **“match_parent”** αντίστοιχα. Στο τέλος του View βλέπουμε τα χαρακτηριστικά **“app:headerLayout ”** και **“app:menu”**, όπου ορίζουμε το ActionBar και το μενού αντίστοιχα.

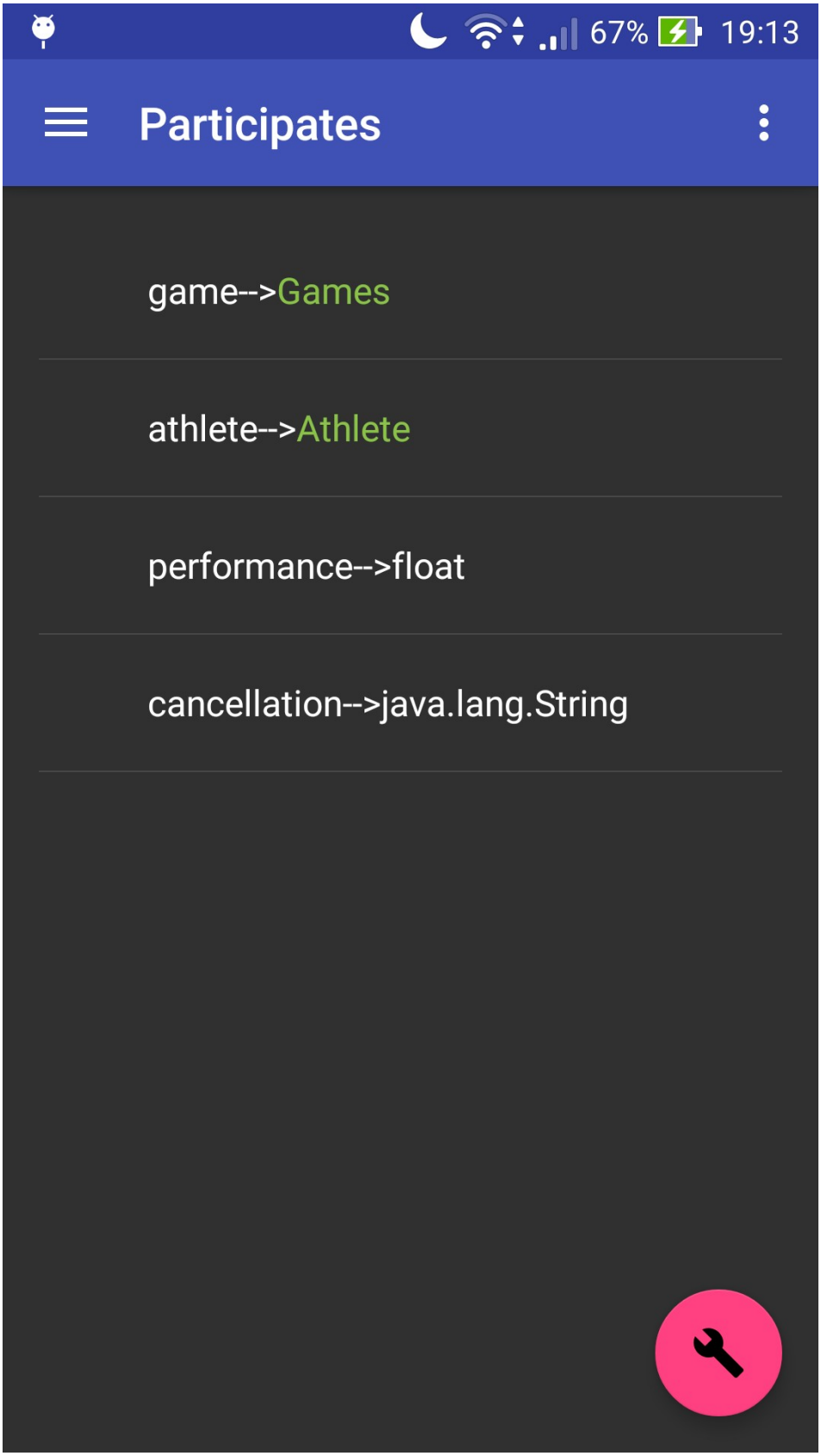
```
1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.v4.widget.DrawerLayout
3.     xmlns:android="http://schemas.android.com/apk/res/android"
4.     xmlns:app="http://schemas.android.com/apk/res-auto"
5.     xmlns:tools="http://schemas.android.com/tools"
6.     android:id="@+id/drawer_layout"
7.     android:layout_width="match_parent"
8.     android:layout_height="match_parent"
9.     android:fitsSystemWindows="true"
10.        tools:openDrawer="start">
11.        <include
12.            layout="@layout/app_bar_initial"
13.            android:layout_width="match_parent"
14.            android:layout_height="match_parent" />
15.        <android.support.design.widget.NavigationView
16.            android:id="@+id/nav_view"
17.            android:layout_width="wrap_content"
18.            android:layout_height="match_parent"
19.            android:layout_gravity="start"
20.            android:fitsSystemWindows="true"
21.            app:headerLayout="@layout/nav_header_initial"
22.            app:menu="@menu/activity_initial_drawer" />
23.    </android.support.v4.widget.DrawerLayout>
```

Εικόνα 110 "Κώδικας του αρχείου activity_initial.xml"

Στο Γραφικό Περιβάλλον της Activity αυτής, Εικόνες 111 και 112, ο χρήστης επιλέγει από τον Drawer την κλάση που επιθυμεί. Όταν επιλέξει την επιθυμητή κλάση τότε εμφανίζεται στο κυρίως παράθυρο μία λίστα με τα πεδία της εκάστοτε κλάσης και τους τύπους δεδομένων αυτών. Πατώντας το FloatingButton ο χρήστης μεταβαίνει στην Activity ConstraintsActivity όπου θα θέσει περιορισμούς χρησιμοποιώντας ως κλάση ρίζα την κλάση που επέλεξε σε αυτή την Activity.



Εικόνα 111 "Γραφικό Περιβάλλον του αρχείου activity_initial.xml (Drawer)"



Εικόνα 112 "Γραφικό Περιβάλλον του αρχείου activity_initial.xml (Λίστα Πεδίων Κλάσης)"

3.3.4 Activity Constraints Layout

Στο αρχείο “activity_constraints.xml” είναι αποθηκευμένα τα Views που χρειαζόμαστε για να αλληεπιδράσει ο κώδικας που είδαμε στο προηγούμενο υποκεφάλαιο με τον χρήστη για να επιλέξει τα πεδία στα οποία θα θέσει τους περιορισμούς. Στις γραμμές ένα έως εννέα, κώδικας της Εικόνας 113, ορίζουμε τα βασικά χαρακτηριστικά του αρχείου οδηγιών για τη δημιουργία του Γραφικού Περιβάλλοντος. Στη συνέχεια δηλώνουμε το root Layout που θα χρησιμοποιηθεί και θα περικλύει όλα τα υπόλοιπα Views, στην συγκεκριμένη περίπτωση επιλέξαμε να χρησιμοποιήσουμε το **“android.support.design.widget.CoordinatorLayout”** το οποίο αποτελεί ένα FrameLayout με υπερ λειτουργίες. Στη συγκεκριμένη περίπτωση χρησιμοποιείται ως ένα container για συγκεκριμένη αλληλεπίδραση με ένα ή περισσότερα παιδιά Views. Οι επόμενες δύο γραμμές ενημερώνουν το Λειτουργικό Σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει τόσο χώρο οριζόντια και κατακόρυφα όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“match_parent”**. Τέλος, έχουμε το View **“android.support.design.widget.FloatingActionButton”**, που είναι το FloatingButton που πατώντας το ο χρήστης εκτελεί το query και μεταβαίνει στην υπεύθυνη Activity για την τύπωση των αποτελεσμάτων, την RecursivePrint.

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.design.widget.CoordinatorLayout
3.     xmlns:android="http://schemas.android.com/apk/res/android"
4.     xmlns:app="http://schemas.android.com/apk/res-auto"
5.     xmlns:tools="http://schemas.android.com/tools"
6.     android:layout_width="match_parent"
7.     android:layout_height="match_parent"
8.     android:fitsSystemWindows="true"
9. tools:context="com.example.finalthesis.db4o_the_project.Constr
   aintsActivity">
10.      <android.support.design.widget.AppBarLayout
11.          android:layout_width="match_parent"
12.          android:layout_height="wrap_content"
13.          android:theme="@style/AppTheme.AppBarOverlay">
14.          <android.support.v7.widget.Toolbar
15.              android:id="@+id/toolbar"
16.              android:layout_width="match_parent"
17.              android:layout_height="?attr/actionBarSize"
18.              android:background="?attr/colorPrimary"
19.              app:popupTheme="@style/AppTheme.PopupOverlay" />
20.      </android.support.design.widget.AppBarLayout>
21.      <include layout="@layout/content_constraints" />
22.      <android.support.design.widget.FloatingActionButton
23.          android:id="@+id/fab"
24.          android:layout_width="wrap_content"
25.          android:layout_height="wrap_content"
26.          android:layout_gravity="bottom|end"
27.          android:layout_margin="@dimen/fab_margin"
28.          android:src="@mipmap/ic_done_all_black_24dp"
29.          android:background="#f9f9f9" />
30.  </android.support.design.widget.CoordinatorLayout>
```

Εικόνα 113 "Κώδικας του αρχείου activity_constraints.xml"

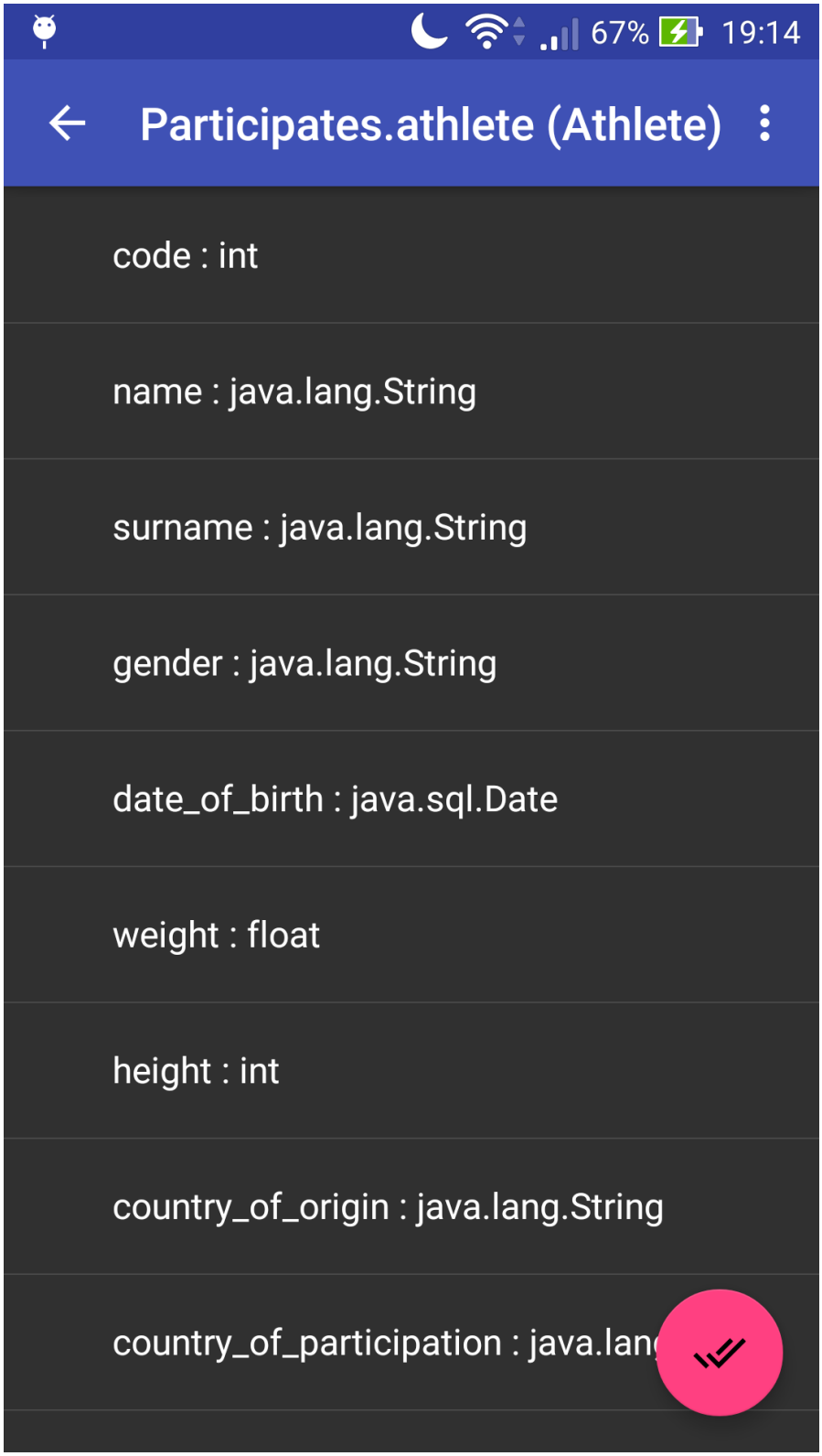
Το αρχείο "content_constraints.xml", κώδικας της Εικόνας 114, είναι το Fragment που περιέχει το RecyclerView και στο οποίο εμφανίζεται η λίστα με τα πεδία στα οποία ο χρήστης καλείται να θέσει περιορισμούς. Στις γραμμές ένα έως εννέα, ορίζουμε τα βασικά χαρακτηριστικά του αρχείου οδηγιών για τη δημιουργία του Γραφικού Περιβάλλοντος. Στην συγκεκριμένη περίπτωση επιλέξαμε να χρησιμοποιήσουμε το Layout **RelativeLayout**, όπου τα αντικείμενα τοποθετούνται στην οθόνη με βάση τη θέση των υπόλοιπων αντικειμένων. Οι επόμενες δύο γραμμές ενημερώνουν το Λειτουργικό Σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει τόσο χώρο οριζόντια και κατακόρυφα όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **"match_parent"**. Η μεταβλητή **"app:layout_behavior"** μας δείχνει ότι μπορούμε σε περίπτωση που έχουμε αρκετά αντικείμενα για προβολή να χρησιμοποιήσουμε ένα Scroller. Στη μεταβλητή **"context"** δηλώνουμε το όνομα της κλάσης με την οποία θα συνδέσουμε το συγκεκριμένο Fragment. Στο τέλος δηλώνουμε ότι το συγκεκριμένο Fragment θα ενσωματωθεί στο Fragment **"activity_constraints"**, που είδαμε

νωρίτερα. Το επόμενο View του Fragment είναι το **“android.support.v7.widget.RecyclerView”**, που φέρει ευθύνη για την προβολή των περιορισμών.

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     xmlns:tools="http://schemas.android.com/tools"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent"
7.
   app:layout_behavior="@string/appbar_scrolling_view_behavior"
8. tools:context="com.example.finalthesis.db4o_the_project.Const
   raintsActivity"
9.     tools:showIn="@layout/activity_constraints">
10.       <android.support.v7.widget.RecyclerView
11.           xmlns:tools="http://schemas.android.com/tools"
12.           android:id="@+id/reflectFieldsRecyclerView"
13.           android:layout_width="match_parent"
14.           android:layout_height="match_parent"
15.           app:layoutManager="LinearLayoutManager"
16.           tools:listitem="@layout/reflect_field_item"/>
17.   </RelativeLayout>
```

Εικόνα 114 “Κώδικας του αρχείου content_constraints.xml”

Στο Γραφικό Περιβάλλον που αποτελεί συνδυασμό των παραπάνω Fragment, Εικόνα 115, ο χρήστης επιλέγει το πεδίο που επιθυμεί να θέσει τον περιορισμό με τη χρήση του onLongClickListener. Όταν επιλέξει πεδίο που αποτελεί αναφορά σε κάποια άλλη κλάση που βρίσκεται στη Βάση Δεδομένων τότε φορτώνεται αυτή η κλάση ώστε να θέσει περιορισμό στα πεδία αυτής. Σε αντίθετη περίπτωση εκκινείται η κλάση ConstraintsDialogFragment για να θέσει ο χρήστης τον επιθυμητό περιορισμό. Πατώντας το FloatingButton ο χρήστης μεταβαίνει στην Activity ResultPrint όπου θα προβληθεί το αποτέλεσμα του ερωτήματος (query).



Εικόνα 115 "Γραφικό Περιβάλλον του Constraints Activity"

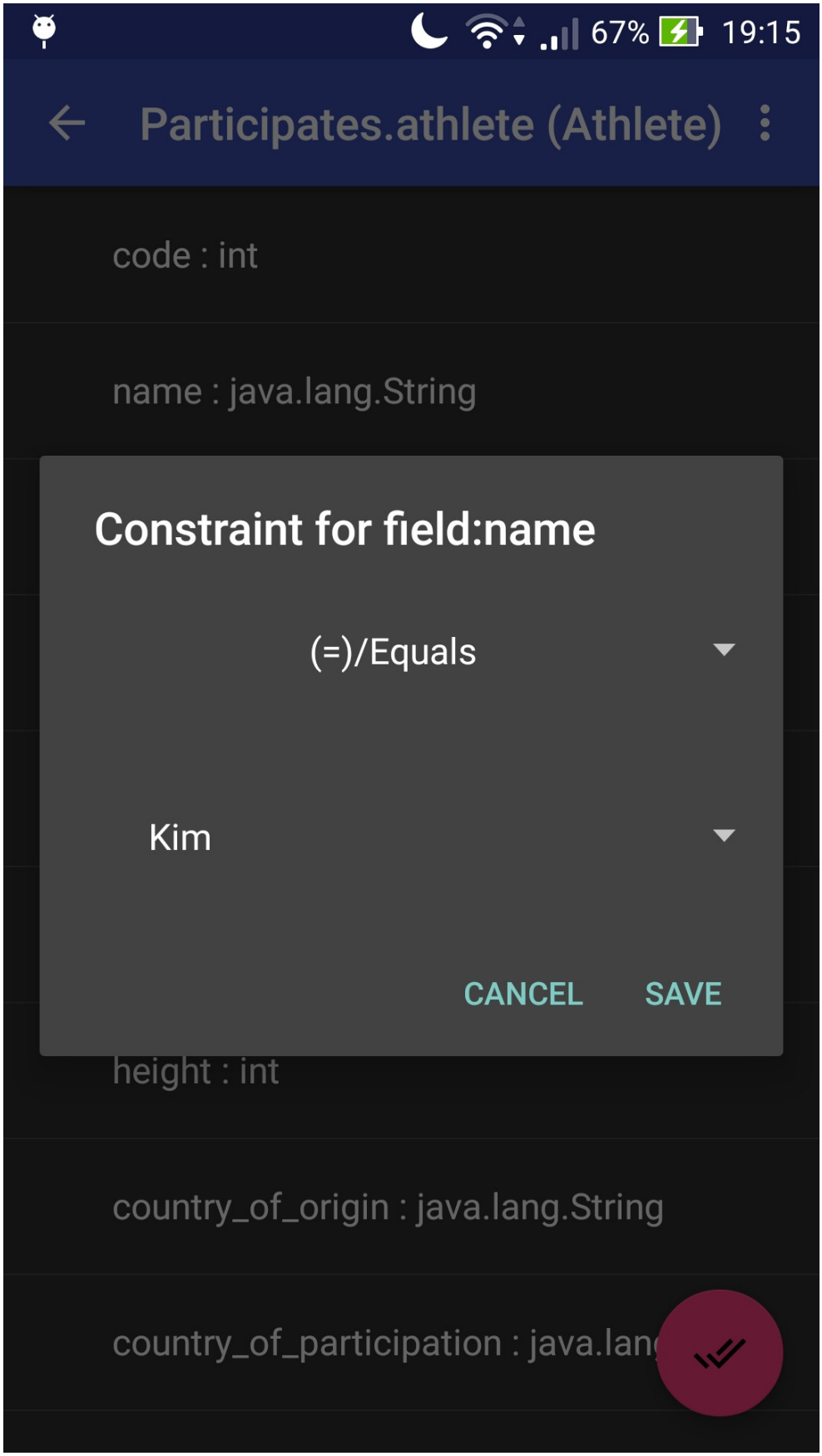
3.3.5 ConstraintDialogFragment Layout

Στο αρχείο “dialog_constraint.xml” είναι αποθηκευμένα τα Views που χρειαζόμαστε για να αλληεπιδράσει ο κώδικας που είδαμε στο προηγούμενο υποκεφάλαιο με τον χρήστη για να θέσει τους επιθυμητούς περιορισμούς, τελεστές και τιμές πεδίων. Στις γραμμές ένα έως εννέα, κώδικας της Εικόνας 116, ορίζουμε τα βασικά χαρακτηριστικά του αρχείου οδηγιών για τη δημιουργία του Γραφικού Περιβάλλοντος. Αυτά είναι με τη σειρά το root Layout που θα χρησιμοποιηθεί και θα περικλύει όλα τα υπόλοιπα Views, στην συγκεκριμένη περίπτωση επιλέξαμε να χρησιμοποιήσουμε το Layout **RelativeLayout**. Οι επόμενες δύο γραμμές ενημερώνουν το Λειτουργικό Σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει τόσο χώρο οριζόντια και κατακόρυφα όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“match_parent”**. Η μεταβλητή **“android:orientation”** καθορίζει ότι τα παιδιά Views του RelativeLayout θα απεικονίζονται κάθετα. Τα παιδιά αυτού είναι τα Spinner, για την εμφάνιση των operator ανάλογα με τον τύπο δεδομένων της εκάστοτε μεταβλητής. Οι επόμενες δύο γραμμές ενημερώνουν το Λειτουργικό Σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει οριζόντια και κατακόρυφα χώρο όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“wrap_content”** και **“50dp”** αντίστοιχα. Το **“50dp”** σημαίνει ότι ανάλογα με την πυκνότητα των πίξελ της εκάστοτε συσκευής θα μεταβάλλεται και το ύψος του Spinner. Οι μεταβλητές **“gravity”** και **“textAlignment”** δηλώνουν ότι το Spinner θα βρίσκεται στο κέντρο του παραθύρου και το κείμενο θα βρίσκεται στο κέντρο του Spinner αντίστοιχα. Στη συνέχεια έχουμε ένα View τύπου EditText με παρόμοια χαρακτηριστικά με το Spinner, όταν η επιλεγμένη μεταβλητή δεν είναι αλφαριθμητικό με τον operator “Equals” ή Boolean, τότε ο χρήστης το χρησιμοποιεί για να θέσει την τιμή του περιορισμού. Όταν η επιλεγμένη μεταβλητή είναι αλφαριθμητικό με τον operator “Equals” ή Boolean, τότε χρησιμοποιούμε το Spinner με **ID= valueSpinner** αντί του EditText, ωστόσο επειδή στις περισσότερες περιπτώσεις χρησιμοποιείται το EditText θέτουμε τη μεταβλητή **“visibility”** του δεύτερου Spinner ως **invisible** έτσι ώστε να μην είναι ορατό στο χρήστη.

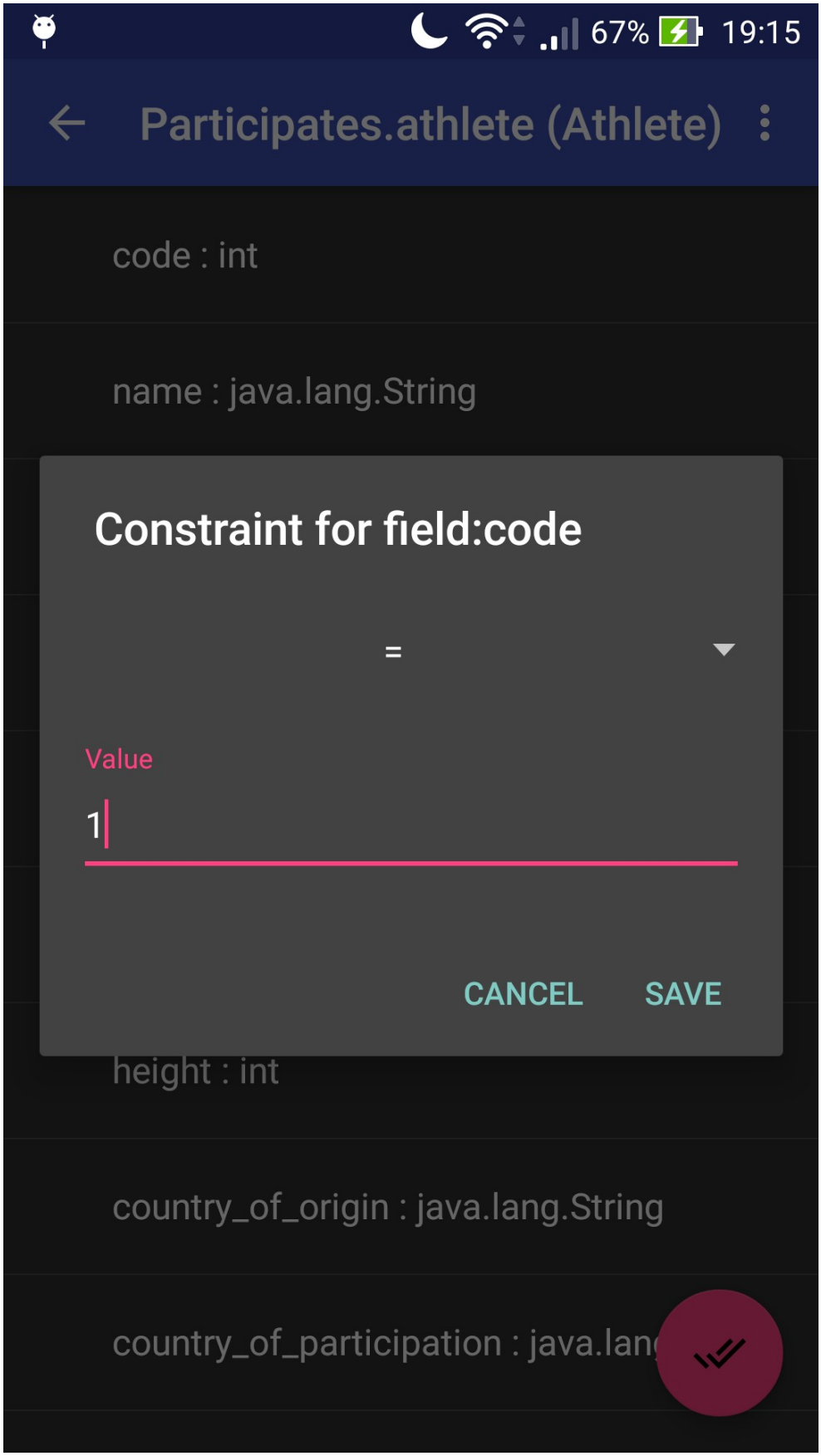
```
1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout
3.     xmlns:android="http://schemas.android.com/apk/res/android"
4.     android:layout_width="match_parent"
5.     android:layout_height="match_parent"
6.     android:orientation="vertical">
7.     <Spinner
8.         android:id="@+id/operatorSpinner"
9.         android:layout_width="match_parent"
10.            android:layout_height="50dp"
11.            android:layout_margin="@dimen/text_margin"
12.            android:gravity="center"
13.            android:textAlignment="center"
14.
    android:textAppearance="?attr/textAppearanceListItem">
15.        </Spinner>
16.        <android.support.design.widget.TextInputLayout
17.            android:id="@+id/valueTextInputLayout"
18.            android:layout_width="match_parent"
19.            android:layout_height="wrap_content"
20.            android:layout_below="@+id/operatorSpinner">
21.            <EditText
22.                android:id="@+id/valueEditText"
23.                android:layout_width="match_parent"
24.                android:layout_height="50dp"
25.                android:layout_margin="@dimen/text_margin"
26.                android:hint="Value"
27.
    android:textAppearance="?attr/textAppearanceListItem"/>
28.        </android.support.design.widget.TextInputLayout>
29.        <Spinner
30.            android:id="@+id/valueSpinner"
31.            android:layout_width="match_parent"
32.            android:layout_height="50dp"
33.            android:layout_below="@+id/operatorSpinner"
34.            android:layout_margin="@dimen/text_margin"
35.            android:gravity="center"
36.            android:textAlignment="center"
37.            android:textAppearance="?attr/textAppearanceListItem"
38.            android:visibility="invisible">
39.        </Spinner>
40.    </RelativeLayout>
```

Εικόνα 116 "Κώδικας του αρχείου dialog_constraints.xml"

Στο συγκεκριμένο Fragment, Εικόνες 117 και 118, βλέπουμε το όνομα του επιλεγμένου πεδίου, ένα Spinner από όπου ο χρήστης θα διαλέξει έναν operator με βάση τον τύπο δεδομένων που είναι η εκάστοτε μεταβλητή. Στη συνέχεια ανάλογα με τον τύπο δεδομένων της επιλεγμένης μεταβλητής καθώς και με τον επιλεγμένο operator εμφανίζουμε το View τύπου Spinner ή EditText, όπου ο χρήστης τελικά επιλέγει ή θέτει τον περιορισμό αντίστοιχα. Τέλος, αν πατήσει αποθήκευση ο περιορισμός θα προστεθεί στην αντίστοιχη λίστα. Στην περίπτωση που πατήσει ακύρωση, τότε ο χρήστης θα μεταφερθεί στην Activity ConstraintsActivity δίχως να ληφθεί υπ' όψιν οποιαδήποτε ενέργεια του χρήστη έως τώρα.



Εικόνα 117 "Γραφικό Περιβάλλον του αρχείου dialog_constraints.xml (Τιμή επιλογής σε Spinner)"



Εικόνα 118 "Γραφικό Περιβάλλον του αρχείου dialog_constraints.xml (Τιμή επιλογής σε EditText)"

3.3.6 Activity RecursivePrint Layout

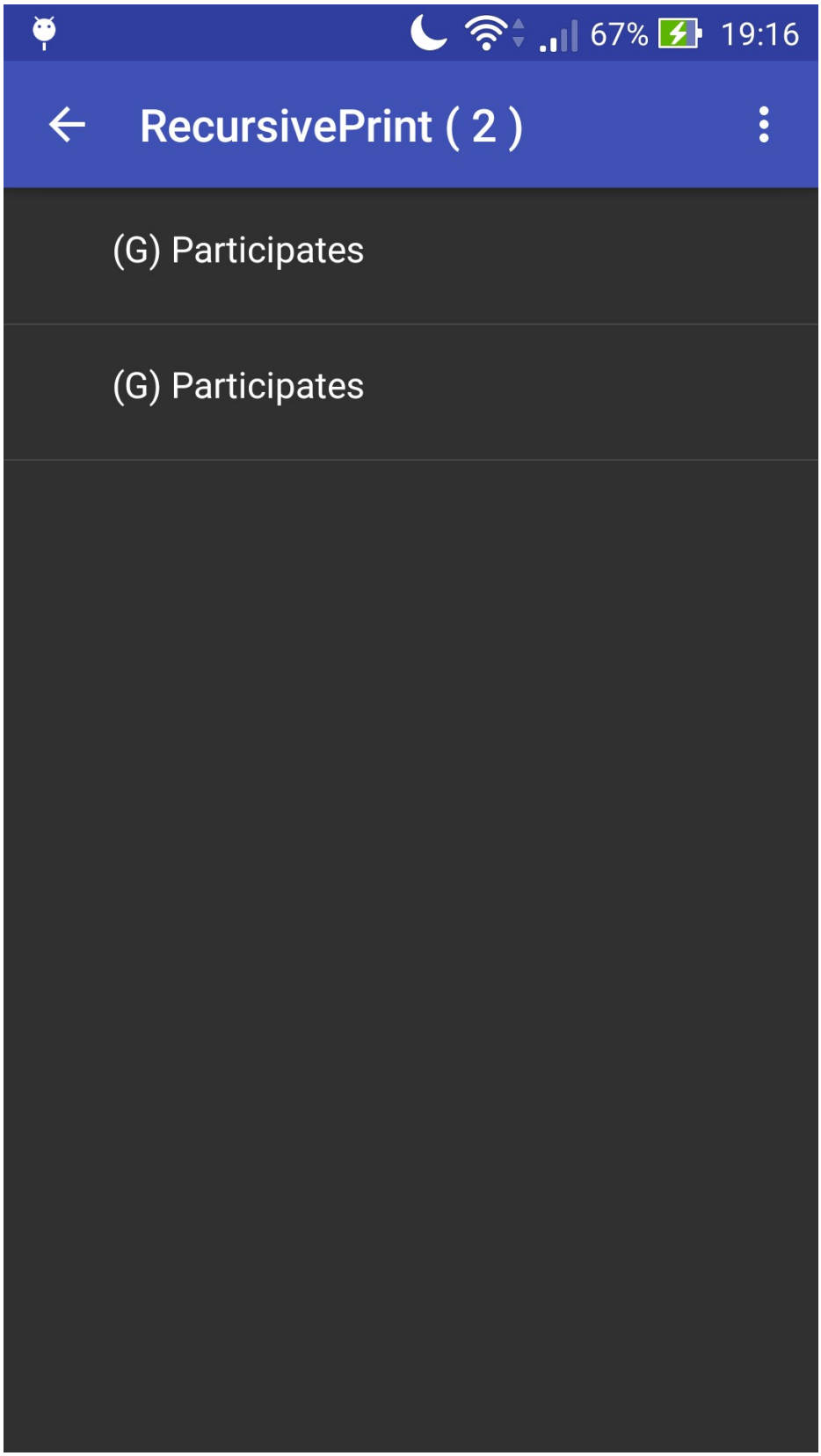
Στο αρχείο “activity_recursive_print.xml” είναι αποθηκευμένα τα Views που χρειαζόμαστε για να προβάλλουμε τα αποτελέσματα του ερωτήματος του χρήστη. Στις γραμμές ένα έως δέκα, κώδικας της Εικόνας 119, ορίζουμε τα βασικά χαρακτηριστικά, αυτά είναι με τη σειρά η δήλωση της έκδοσης XML που χρησιμοποιούμε καθώς και η επιθυμητή κωδικοποίηση. Στη συνέχεια έχουμε το root Layout που θα χρησιμοποιηθεί και θα περικλύει όλα τα υπόλοιπα Views, στην συγκεκριμένη περίπτωση επιλέξαμε να χρησιμοποιήσουμε το **“android.support.v4.widget.DrawerLayout”**. Οι επόμενες δύο γραμμές ενημερώνουν το Λειτουργικό Σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει τόσο χώρο οριζόντια και κατακόρυφα όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“match_parent”**. Στη συνέχεια το tag **“include”** όπου ορίζουμε ότι θα πρέπει να καταλαμβάνει τόσο χώρο οριζόντια και κατακόρυφα όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“match_parent”** και θα εναποθέσουμε σε αυτό ένα Fragment το οποίο θα περιέχει ένα View τύπου ReclyclerView και θα εμφανίζει τα αποτελέσματα του ερωτήματος (query) με τη χρήση λίστας.

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.v4.widget.DrawerLayout
3.     xmlns:android="http://schemas.android.com/apk/res/android"
4.     xmlns:app="http://schemas.android.com/apk/res-auto"
5.     xmlns:tools="http://schemas.android.com/tools"
6.     android:id="@+id/drawer_layout"
7.     android:layout_width="match_parent"
8.     android:layout_height="match_parent"
9.     android:fitsSystemWindows="true"
10.        >
11.        <include
12.            layout="@layout/app_bar_recursive_print"
13.            android:layout_width="match_parent"
14.            android:layout_height="match_parent" />
15.    </android.support.v4.widget.DrawerLayout>
```

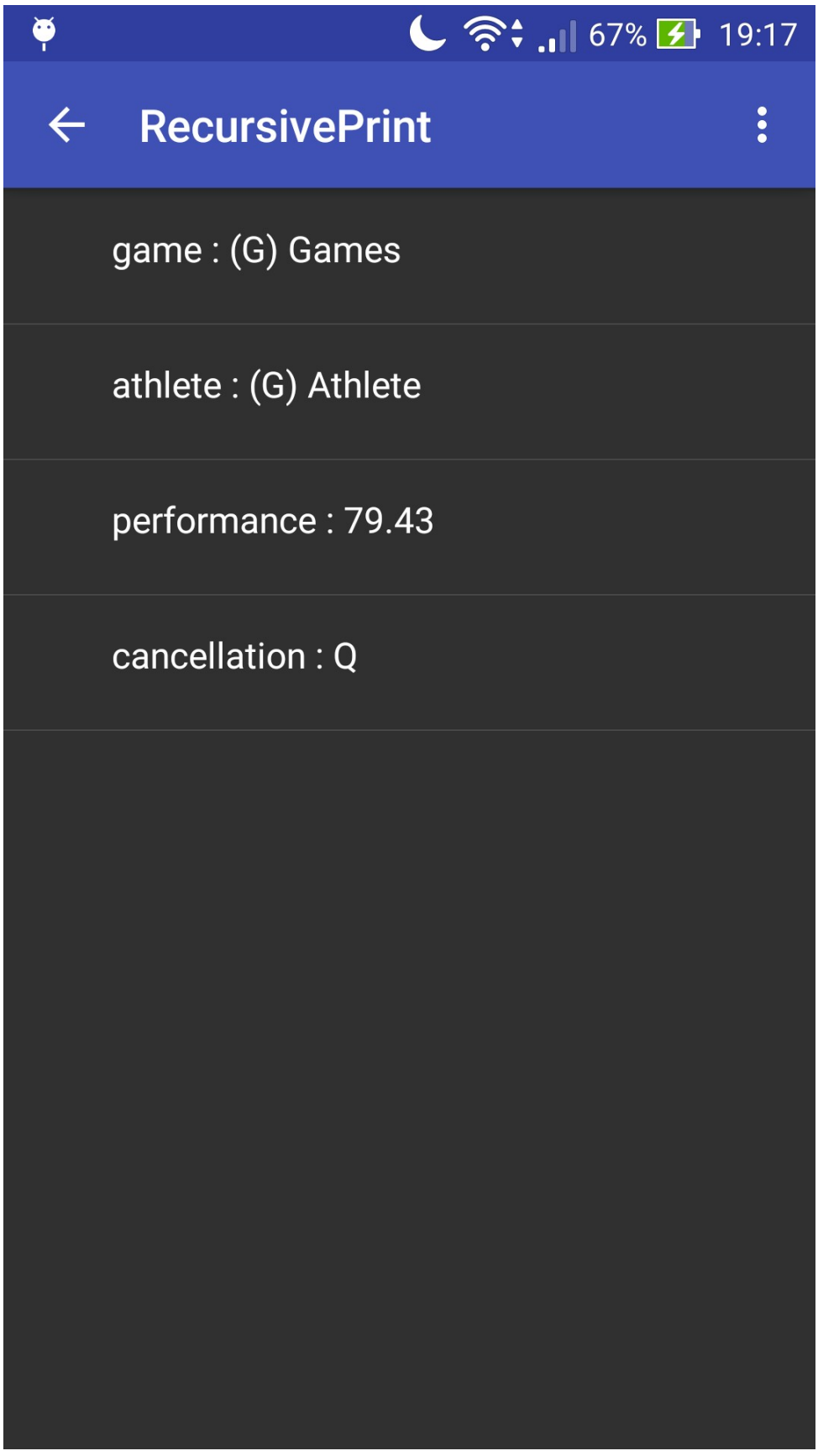
Εικόνα 119 “Κώδικας του αρχείου activity_recursive_print.xml”

Στο συγκεκριμένο Fragment, Εικόνες 120 και 121, ο χρήστης θα δεί τα αντικείμενα που πληρούν τους περιορισμούς που έχει θέσει. Στην αρχή προβάλουμε όλα τα αντικείμενα τύπου κλάσης ρίζα που πληρούν τους επιθυμητούς περιορισμούς. Στη συνέχεια ο χρήστης επιλέγοντας ένα από αυτά μπορεί να δεί τις τιμές των πεδίων του αντικειμένου αυτού. Στην περίπτωση που επιλέξει να δει πεδίο το οποίο

αποτελεί αναφορά σε άλλο αντικείμενο τότε ανοίγουμε ένα νέο παράθυρο που αυτή τη φορά προβάλλει τις τιμές των πεδίων αυτού του αντικειμένου.



Εικόνα 120 “Γραφικό Περιβάλλον του αρχείου activity_recursive_print.xml (Επιλογή Αρχικού Αντικειμένου)”



Εικόνα 121 "Γραφικό Περιβάλλον του αρχείου activity_recursive_print.xml (Προβολή Τιμών Πεδίων Αντικειμένου)"

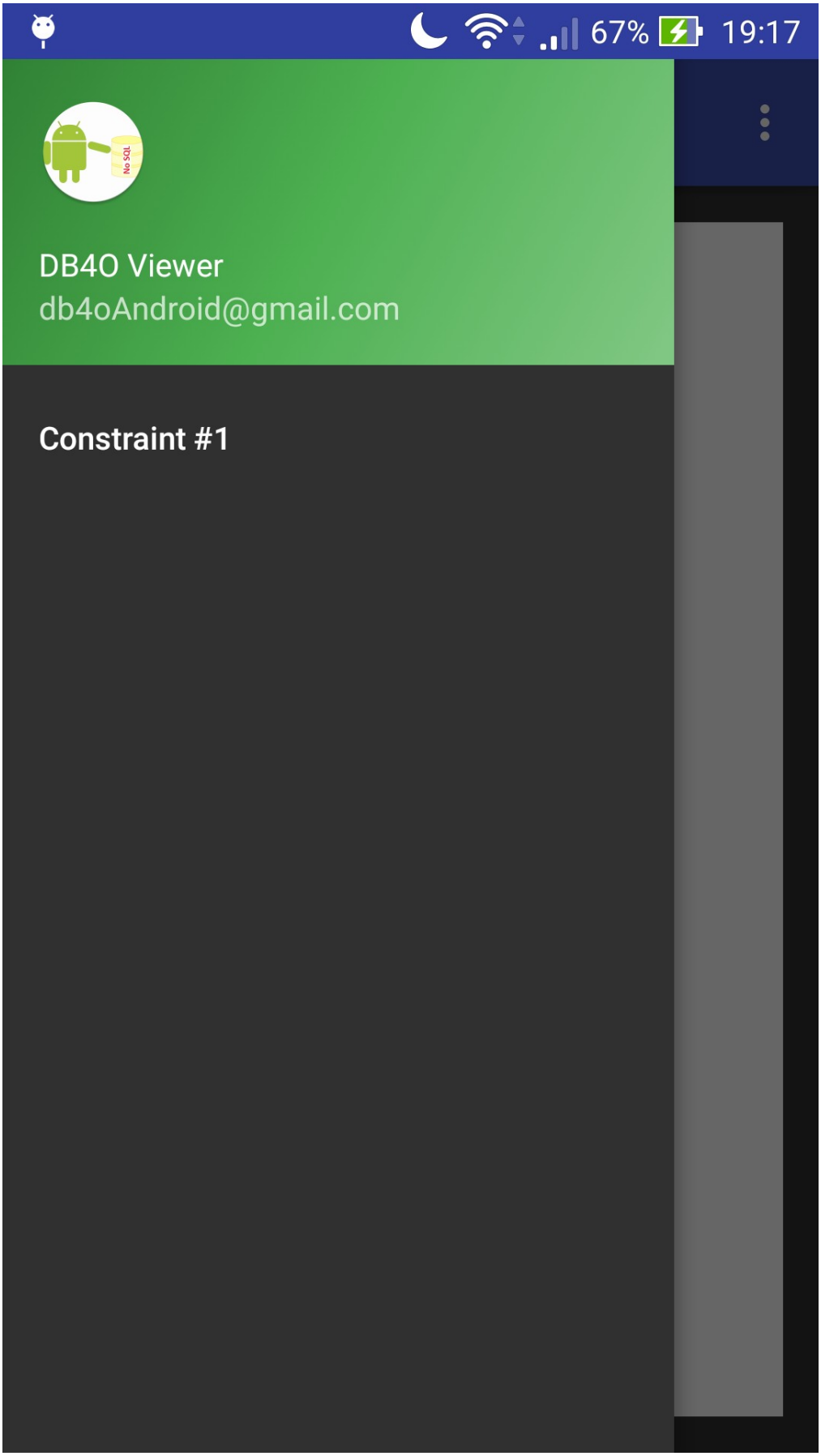
3.3.8 Activity WatchMyConstraints Layout

Στο αρχείο “activity_watch_my_constraints.xml” είναι αποθηκευμένα τα Views που χρειαζόμαστε για να αλληλεπιδράσει ο κώδικας που είδαμε στο προηγούμενο υποκεφάλαιο με τον χρήστη για να προβάλλει τους περιορισμούς που έχει κατασκευάσει ο ίδιος. Στις γραμμές ένα έως οχτώ, κώδικας της Εικόνας 122, ορίζουμε τα βασικά χαρακτηριστικά του αρχείου οδηγιών για τη δημιουργία του Γραφικού Περιβάλλοντος. Στη συνέχεια ορίζουμε το root Layout που θα χρησιμοποιηθεί και θα περικλύει όλα τα υπόλοιπα Views, στην συγκεκριμένη περίπτωση επιλέξαμε να χρησιμοποιήσουμε το **“android.support.v4.widget.DrawerLayout”**. Οι επόμενες δύο γραμμές ενημερώνουν το Λειτουργικό Σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει τόσο χώρο οριζόντια και κατακόρυφα όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“match_parent”** που σημαίνει ότι θα καταλάβει τον ίδιο χώρο που διαθέτει το αρχικό View της εφαρμογής. Η μεταβλητή **“android:fitsSystemWindows”** σημαίνει ότι το background του Drawer όταν τον ανοίγουμε θα είναι διαφανές. Με τη χρήση της μεταβλητής **“tools:openDrawer="start"”** δηλώνουμε στο Λειτουργικό Σύστημα ότι θέλουμε να έχουμε πρόσβαση στον Drawer. Στη συνέχεια στο tag **“include”** ορίζουμε ότι θα πρέπει να καταλαμβάνει τόσο χώρο οριζόντια και κατακόρυφα όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“match_parent”**. Στη συγκεκριμένη περίπτωση θα τοποθετήσουμε σε αυτό το κομμάτι το αρχείο “app_bar_watch_my_constraints.xml”. Τέλος, το View **“android.support.design.widget.NavigationView”** είναι το View του Drawer όπου θα εμφανίζονται οι περιορισμοί και θα βρίσκεται στην αριστερή πλευρά και θα εμφανίζεται καθ’ επιλογή του χρήστη. Οι επόμενες δύο γραμμές ενημερώνουν το Λειτουργικό Σύστημα Android ότι το συγκεκριμένο View θα πρέπει να καταλαμβάνει οριζόντια και κατακόρυφα χώρο όσο αναφέρει η τιμή των πεδίων αυτών, στη συγκεκριμένη περίπτωση είναι **“wrap_content”** και **“match_parent”** αντίστοιχα. Στο τέλος του View βλέπουμε τα χαρακτηριστικά **“app:headerLayout ”** και **“app:menu”**, όπου ορίζουμε το ActionBar και το μενού αντίστοιχα.

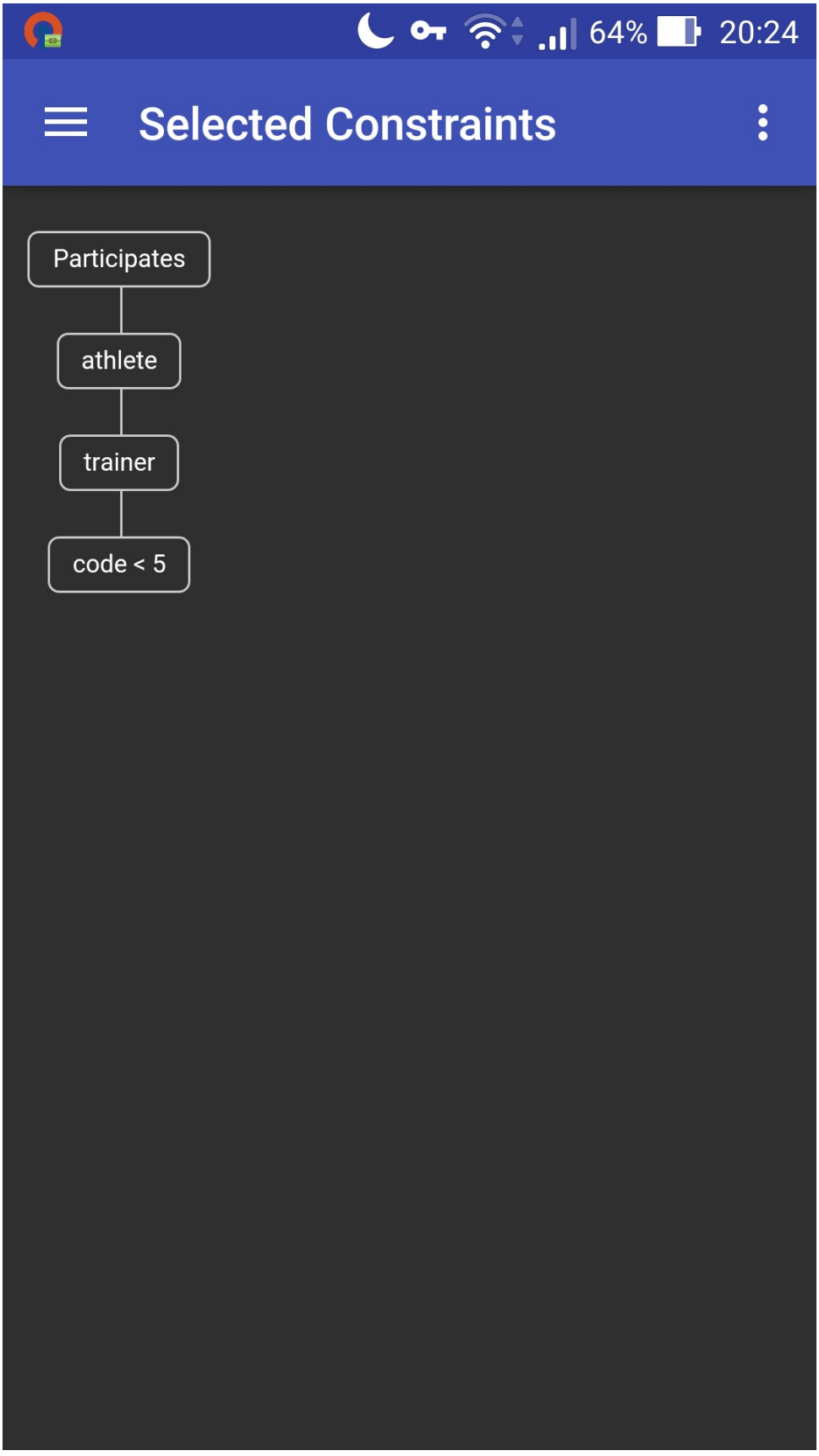
```
1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.v4.widget.DrawerLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     xmlns:tools="http://schemas.android.com/tools"
5.     android:id="@+id/drawer_layout"
6.     android:layout_width="match_parent"
7.     android:layout_height="match_parent"
8.     android:fitsSystemWindows="true"
9.     tools:openDrawer="start">
10.        <include
11.            layout="@layout/app_bar_watch_my_constraints"
12.            android:layout_width="match_parent"
13.            android:layout_height="match_parent" />
14.        <android.support.design.widget.NavigationView
15.            android:id="@+id/nav_viewQ"
16.            android:layout_width="wrap_content"
17.            android:layout_height="match_parent"
18.            android:layout_gravity="start"
19.            android:fitsSystemWindows="true"
20.
   app:headerLayout="@layout/nav_header_watch_my_constraints"
21.
   app:menu="@menu/activity_watch_my_constraints_drawer" />
22.    </android.support.v4.widget.DrawerLayout>
```

Εικόνα 122 "Κώδικας του αρχείου activity_watch_my_constraints.xml"

Στο συγκεκριμένο Fragment, Εικόνες 123 και 124, ο χρήστης επιλέγει από τον Drawer τον επιθυμητό περιορισμό, κάθε περιορισμός εμφανίζεται με τη σειρά που έχει δηλωθεί από τον χρήστη. Όταν ο χρήστης επιλέξει κάποιον από τους περιορισμούς που βρίσκονται στη λίστα, τότε εμφανίζεται στο κυρίως παράθυρο ο περιορισμός που έχει θέσει ο χρήστης με τη χρήση HTML και CSS μέσω ενός αντικειμένου τύπου WebView.



Εικόνα 123 "Γραφικό Περιβάλλον του αρχείου activity_watch_my_constraints.xml (Drawer)"



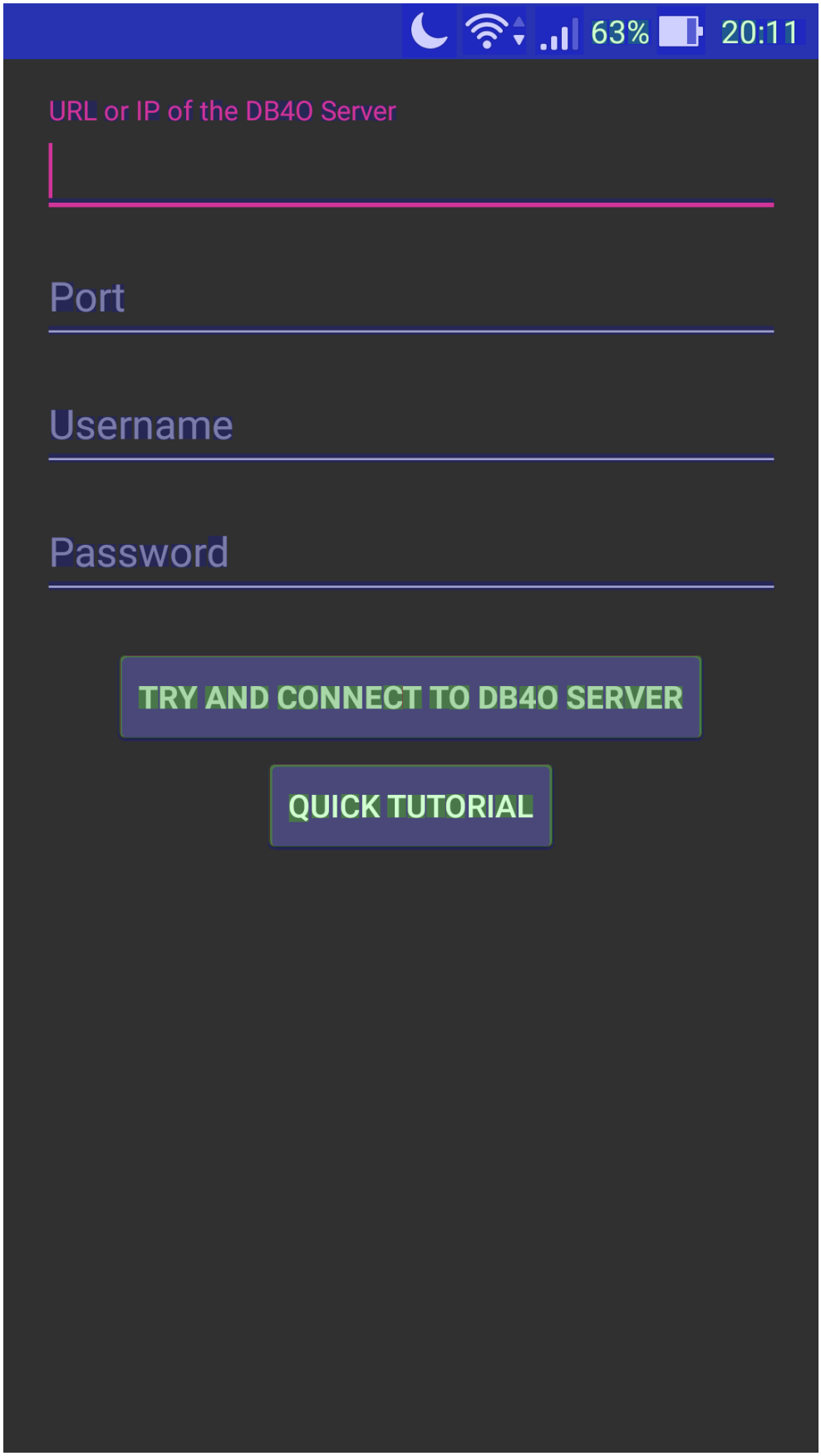
Εικόνα 124 "Γραφικό Περιβάλλον του αρχείου activity_watch_my_constraints.xml (WebView)"

3.3.9 Overdraw Test

Βασικό ρόλο στην ανάπτυξη μίας εφαρμογής για φορητές συσκευές περιορισμένων πόρων διαδραματίζει η σχεδίαση του Γραφικού Περιβάλλοντος χρήστη. Κατά τη διάρκεια της ανάπτυξης της εφαρμογής θελήσαμε να περιορίσουμε όσο το δυνατόν περισσότερο το footprint της εφαρμογής στην χρήση των πόρων της συσκευής. Αυτό το πέτυχαμε τόσο από την αποδοτικότερη χρήση των πόρων της συσκευής προγραμματιστικά, μνήμη, διάρκεια χρήσης κεντρικής μονάδας επεξεργασίας, δικτυακοί πόροι, όσο και στο Γραφικό Περιβάλλον με τη χρήση όσο το δυνατότερο λιγότερων pixel γίνεται. Με τον τρόπο αυτό μειώσαμε σημαντικά το ποσοστό ενέργειας που απαιτείται για την εκτέλεση της εφαρμογής [21]. Ακόμα, για την ανάπτυξη και συνεχή βελτίωση του Γραφικού Περιβάλλοντος καθιστώντας το όσο το δυνατόν πιο αποδοτικό, σε κάθε στάδιο της ανάπτυξης διεξήγαμε το GPU Overdraw Test με τη χρήση του εργαλείου επαναχρησιμοποίησης των pixel που περιλαμβάνεται στο native εργαλείο αποσφαλμάτωσης που προσφέρει το Λειτουργικό Σύστημα Android. Τα αποτελέσματα φαίνονται στις Εικόνες 125 έως 133 και σύμφωνα με τον οδηγό [22]:

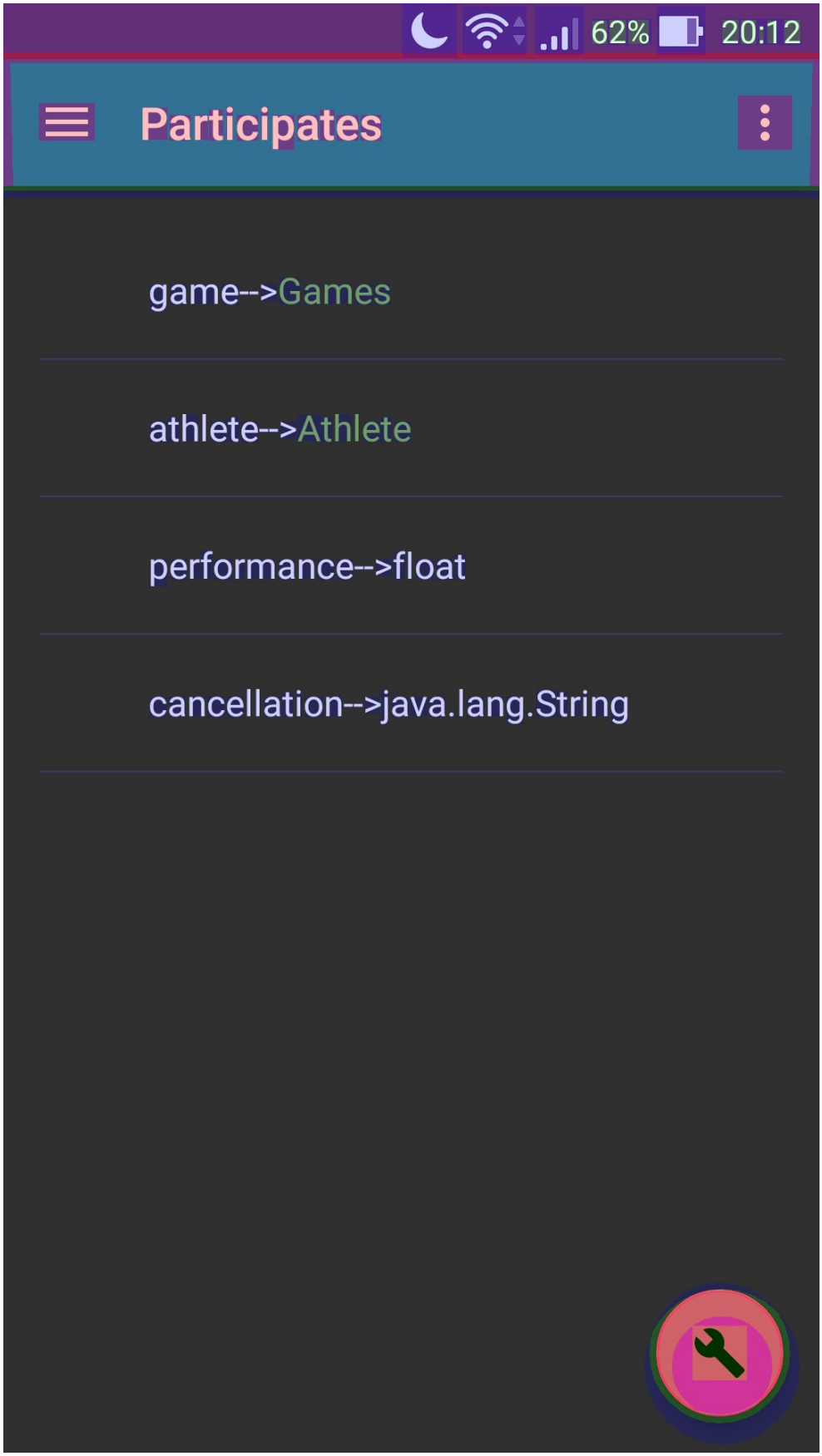
⇒ Το ***φυσικό*** χρώμα δηλώνει δεν έχει γίνει αλληλοεπικάλυψή.
⇒ Το ***μπλε*** σημαίνει ότι έχει γίνει αλληλοεπικάλυψη μία φορά.
⇒ Το ***πράσινο*** σημαίνει ότι έχει γίνει αλληλοεπικάλυψη δύο φορές.
⇒ Το ***ροζ*** σημαίνει ότι έχει γίνει αλληλοεπικάλυψη τρείς φορές.
⇒ Το ***κόκκινο*** σημαίνει ότι έχει γίνει αλληλοεπικάλυψη τέσσερεις ή παραπάνω φορές.

Όπως μπορούμε να δούμε σε κάθε Activity έχουμε πετύχει το καλύτερο δυνατό αποτέλεσμα. Πιο συγκεκριμένα σε όλες τις Activity έχουμε είτε το φυσικό χρώμα είτε το μπλε χρώμα, με σπάνιο φαινόμενο το πράσινο χρώμα, που σημαίνει ότι έχουμε πετύχει έως μία και σε μερικές περιπτώσεις έως δύο επικαλύψεις. Το μόνο στοιχείο των Activity που έχει ροζ χρώμα, αλληλοεπικάλυψη έως τρείς φορές είναι το ActionBar και πιο συγκεκριμένα στις Activity Constraints και RecursivePrint, όπου χρησιμοποιούμε το ActionBar για την προβολή πληροφοριών που αφορούν τα αντικείμενα που αναπαρηστούνται στις Activity αυτές.

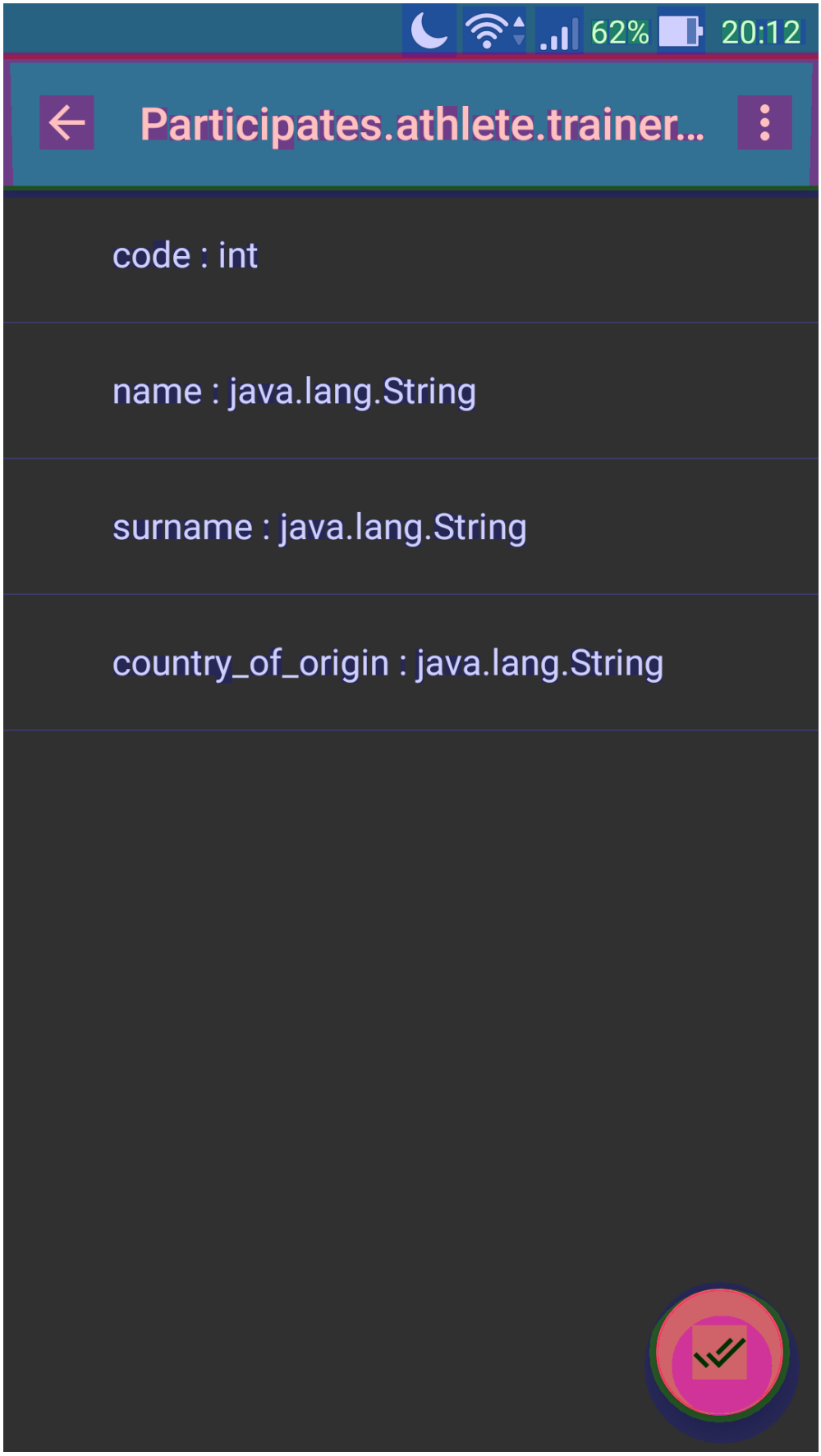


Εικόνα 125 "GPU Overdraw Test (Login Page)"

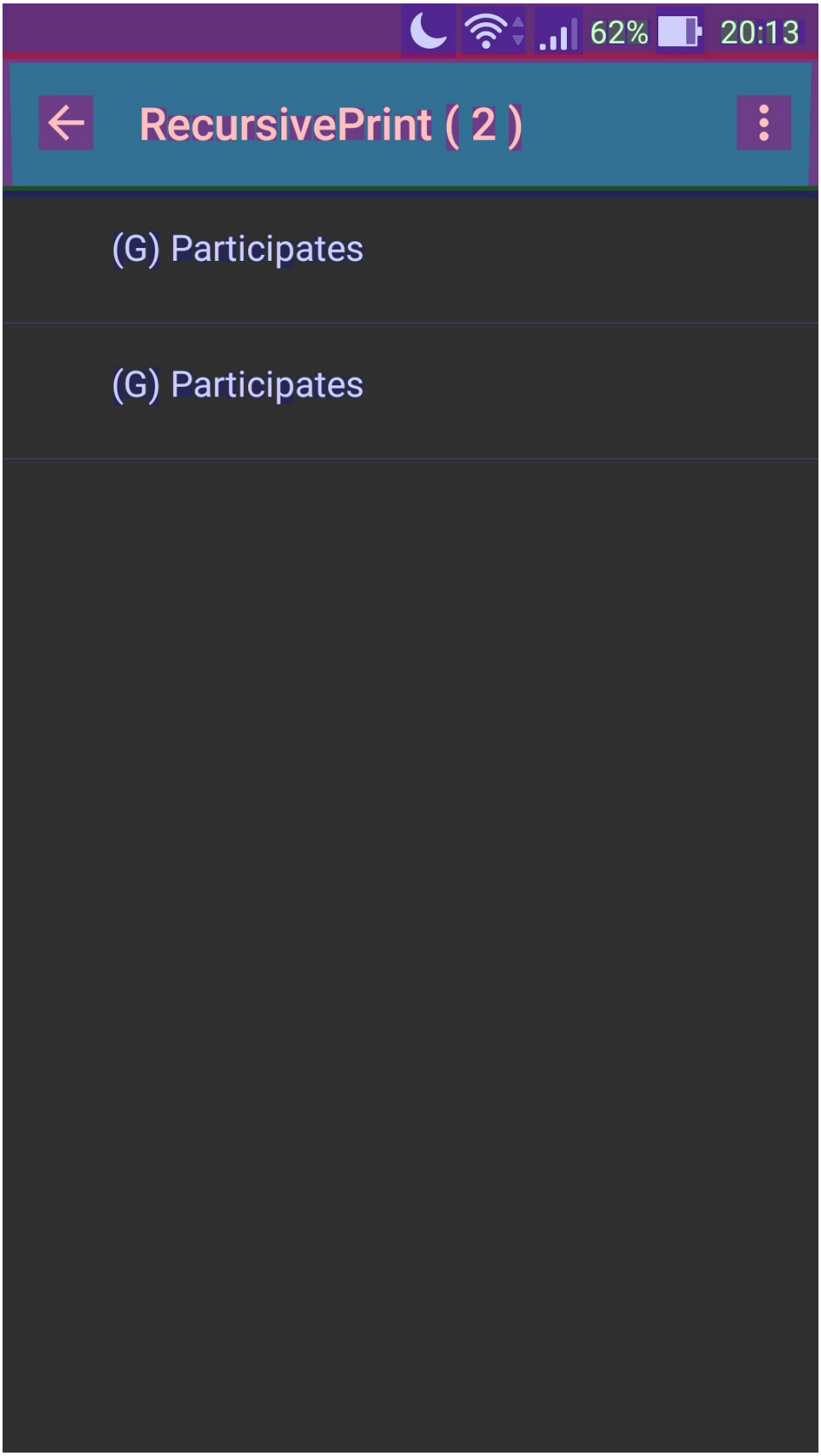
Εικόνα 126 "GPU Overdraw Test (Initial Page, Drawer)"



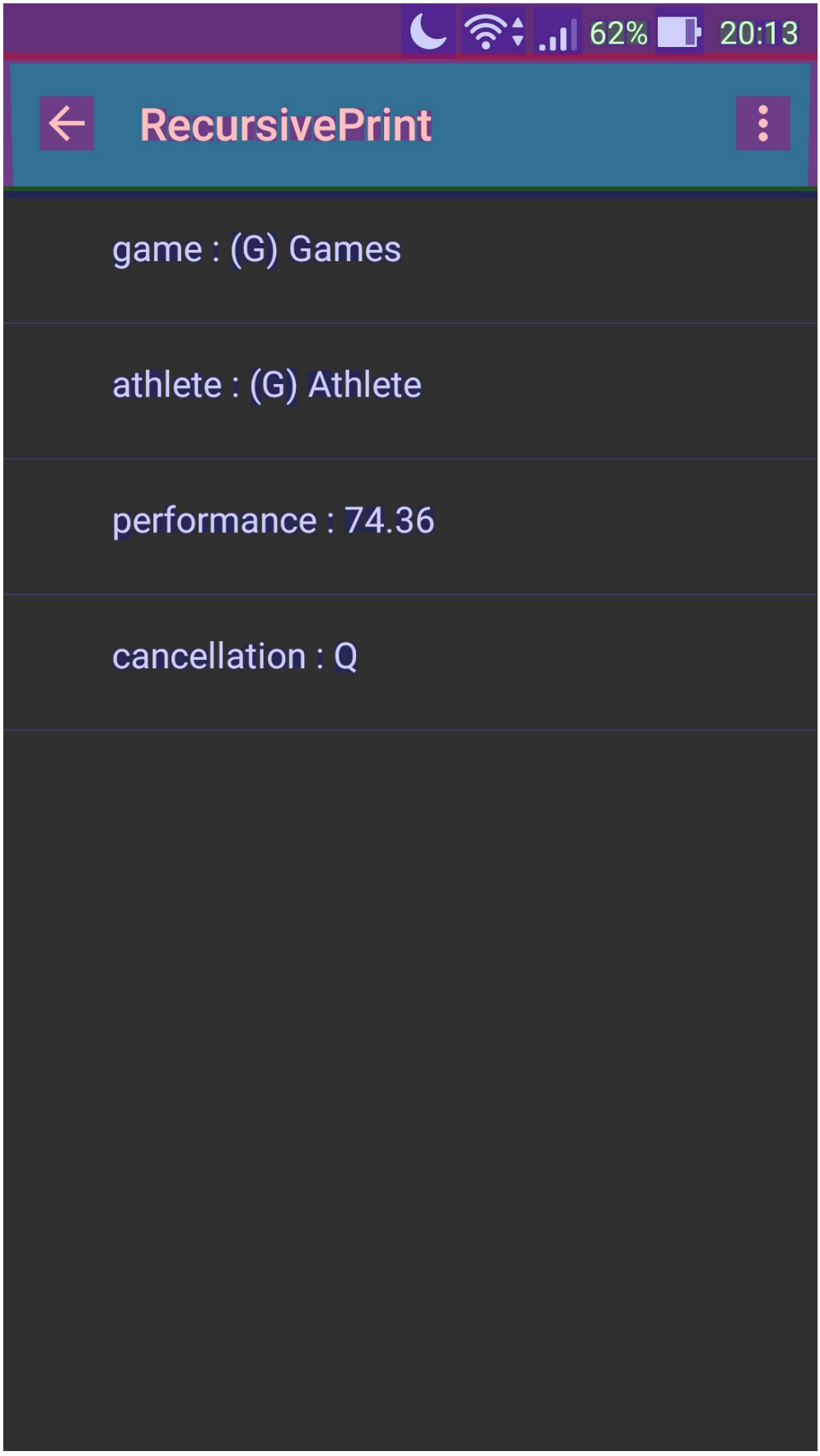
Εικόνα 127 "GPU Overdraw Test (Initial Page, Λίστα Πεδίων)"



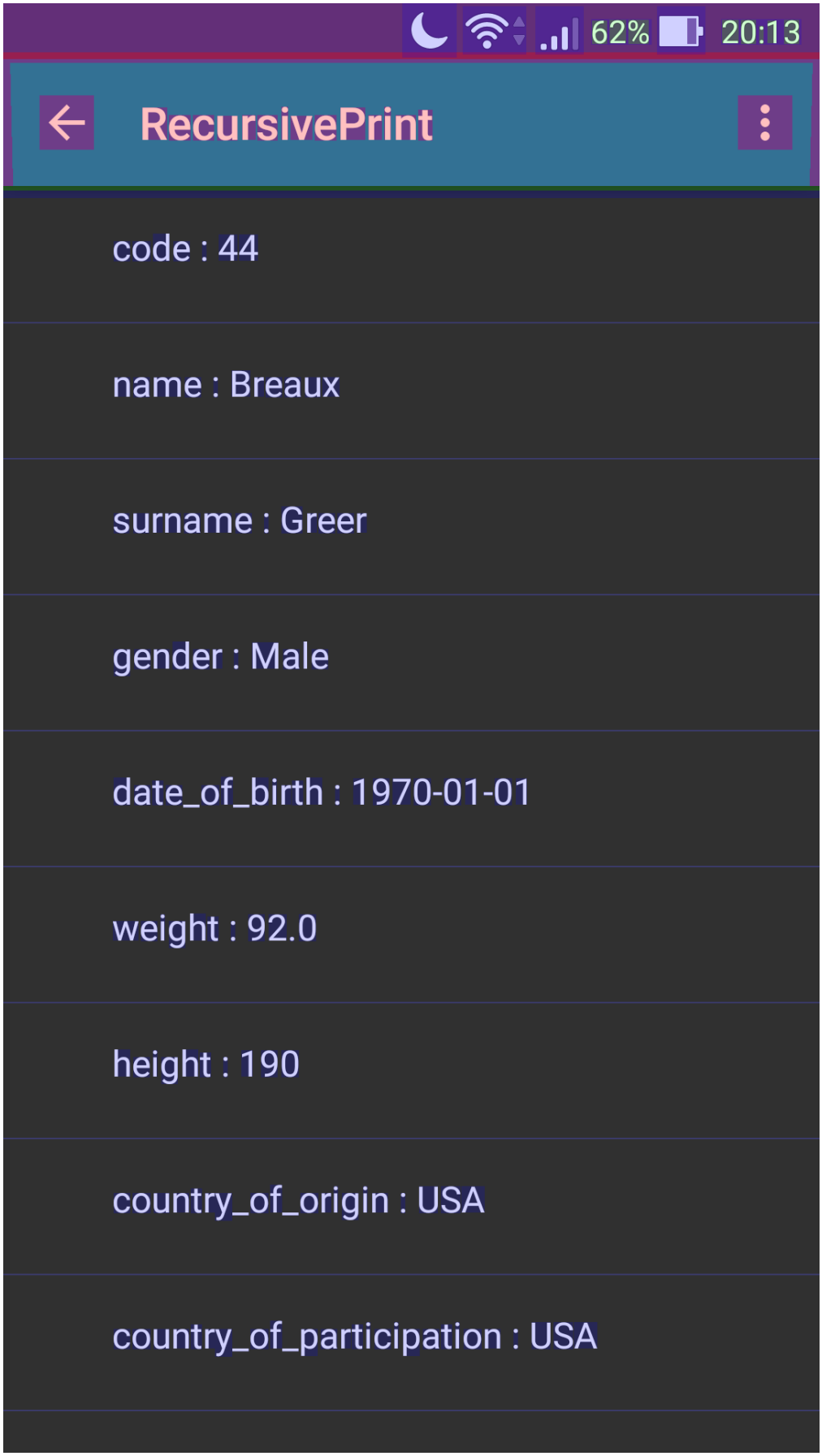
Εικόνα 128 "GPU Overdraw Test (Constraints Page, Μετά το άνοιγμα δύο αντικειμένων)"



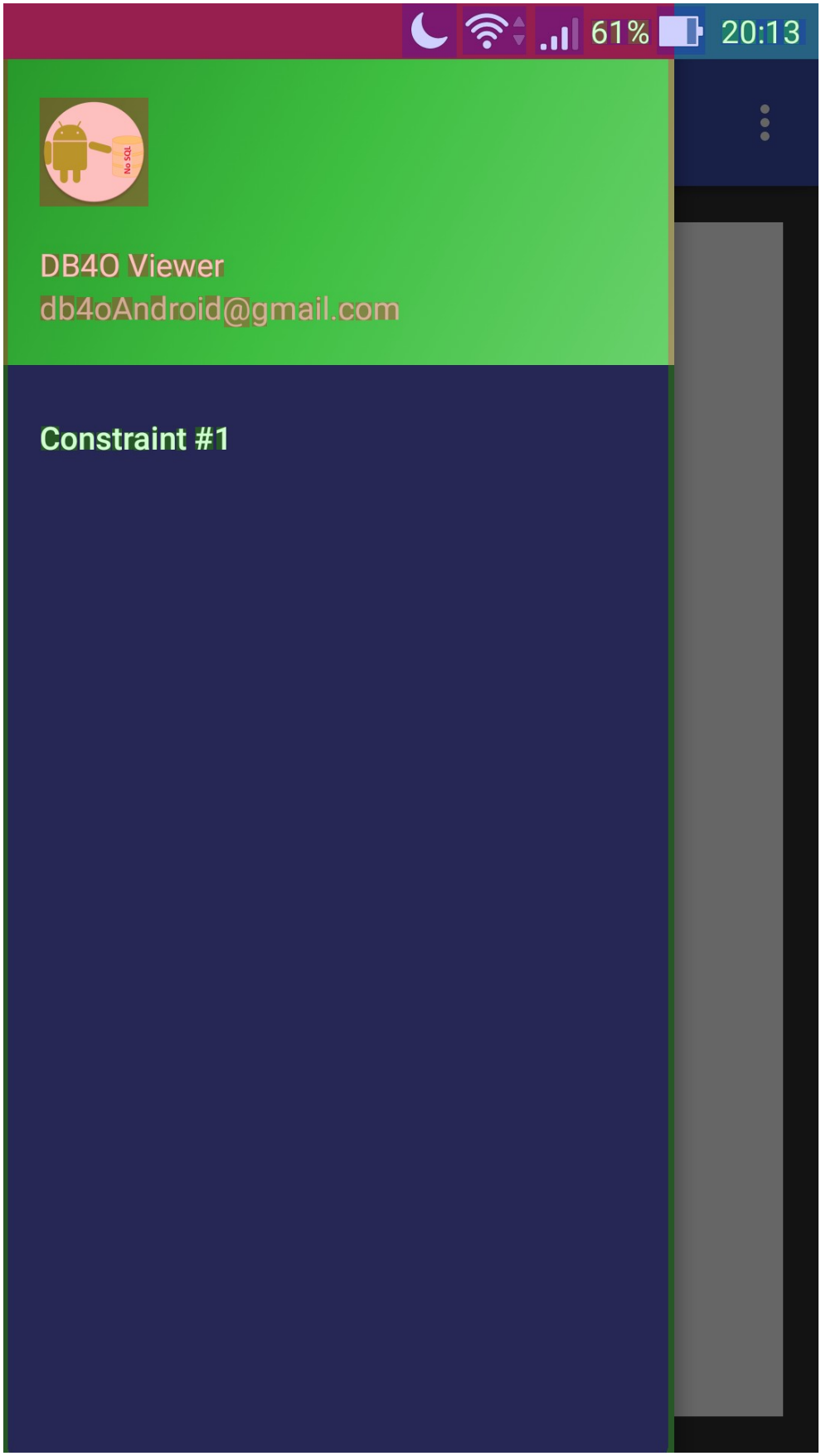
Εικόνα 129 “GPU Overdraw Test (Recursive Print Page, Αρχική επιλογή αντικειμένων)”



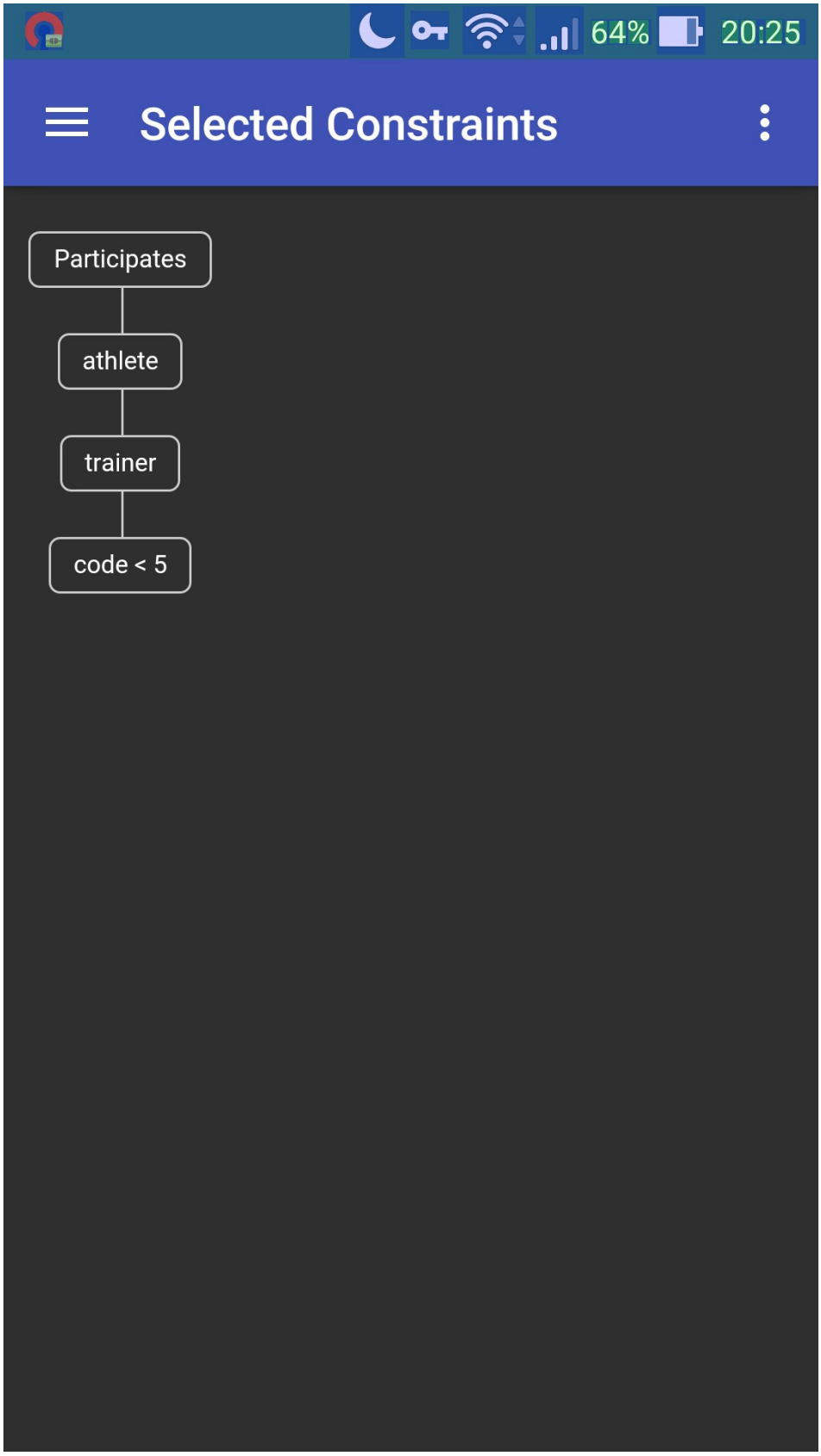
Εικόνα 130 "GPU Overdraw Test (Recursive Print Page, Προβολή Αντικειμένου σε βάθος ΕΝΑ)"



Εικόνα 131 "GPU Overdraw Test (Recursive Print Page, Προβολή Αντικειμένου σε βάθος ΔΥΟ)"



Εικόνα 132 "GPU Overdraw Test (Watch My Constraints Page, Drawer)"



Εικόνα 133 "GPU Overdraw Test (Watch My Constraints Page, WebView)"

ΕΠΙΛΟΓΟΣ

Στο κεφάλαιο που προηγήθηκε έγινε εκτενής αναφορά στα χαρακτηριστικά της εφαρμογής μας. Αφού είδαμε τις βοηθητικές κλάσεις στο πρώτο. Στη συνέχεια στο δεύτερο υποκεφάλαιο είδαμε το κώδικα για την υλοποίηση κάθε Activity ξεχωριστά και στο τρίτο υποκεφάλαιο τα πιο σημαντικά αρχεία που αφορούσαν το Γραφικό Περιβάλλον της εφαρμογής μας τμηματοποιημένα ως προς τις αντίστοιχες κλάσεις του υποκεφαλαίου δύο. Στη συνέχεια θα δούμε τα προβλήματα που προέκυψαν και πως επιλύθηκαν καθώς και μελλοντικά χαρακτηριστικά της εφαρμογής.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Κατά τη διάρκεια εκπόνησης αυτής της πτυχιακής βρεθήκαμε αντιμέτωποι με μία πληθώρα εμποδίων και διλλημάτων, ορισμένα καταφέραμε να τα ξεπεράσουμε(υπερπηδήσουμε), ενώ άλλα όχι. Το πρώτο δίλλημα που αντιμετωπίσαμε ήταν αν θα έπρεπε η εφαρμογή να χρησιμοποιεί ως είσοδο τοπικά αποθηκευμένη στη συσκευή Βάση Δεδομένων τύπου db4o ή μια απομακρυσμένη δικτυακά προσπελάσιμη με τη χρήση ενός εξυπηρετητή. Στην αρχή και μετά από πληθώρα δοκιμών ο χρόνος απόκρισης σε ερωτήματα στην τοπικά αποθηκευμένη ήταν δραματικά μικρότερος σε σχέση με την απομακρυσμένη Βάση Δεδομένων. Ωστόσο, θα θυσιάζαμε τη δυνατότητα του χρήστη να μπορεί να χρησιμοποιήσει μια πληθώρα δικτυακών Βάσεων Δεδομένων τύπου db4o και ορίζοντας ως περιορισμό για τη χρήση της εφαρμογής την ύπαρξη ενός αρχείου db4o στην συσκευή. Με την ολοένα αυξανόμενη δημοτικότητα των εφαρμογών τύπου Πελάτη - Εξυπηρετητή αυτό θα περιόριζε το εύρος του κοινού στο οποίο θα απευθυνόταν. Η λύση στην οποία καταφύγαμε ήταν η δημιουργία ενός τοπικού δικτύου για την δοκιμή της εφαρμογής μας με σκοπό την αποφυγή της καθυστέρησης που οφείλεται στις διαδικασίες δρομολόγησης και επεξεργασίας των πακέτων. Στη συνέχεια καθώς δεν υφίσταται κάποιο πρόγραμμα Εξυπηρετητή προχωρήσαμε στην κατασκευή ενός προγράμματος που θα λειτουργούσε ως Εξυπηρετητής για αρχεία τύπου db4o. Κατά τη δημιουργία του προγράμματος αυτού λάβαμε υπόψιν το γεγονός ότι ένα αρχείο db4o είναι πολύ μικρό σε μέγεθος και δεν καταναλώνει σημαντικό μέρος από τους πόρους ενός συστήματος, πράγμα που αποτελεί ένα από τα πλεονεκτήματα της db4o έναντι σε άλλα Συστήματα Βάσεων Δεδομένων ακόμα και από αυτά τεχνολογίας NoSQL. Αυτό επιτεύχθηκε με τη χρήση της γλώσσας προγραμματισμού Java και ιδίως με το εξελιγμένο και πλήρως παραμετροποιήσιμο εργαλείο που διαθέτει, τον Garbage Collector, ο οποίος μπορεί να διαχειριστεί με άψογο τρόπο την μνήμη του συστήματος απαιτώντας την ύπαρξη μόλις λίγων MB για την εκτέλεση ενός προγράμματος εξυπηρετητή. Η μεγαλύτερη πρόκληση, εμπόδιο, που αντιμετωπίσαμε ήταν η σχεδίαση ενός Γραφικού Περιβάλλοντος για την αναπαράσταση της πληροφορίας, τόσο για τη δημιουργία ενός ερωτήματος με τον ορισμό περιορισμών από τον χρήστη όσο και η απεικόνιση των αποτελεσμάτων με αποδοτικό τρόπο. Η μη γνώση του μεγέθους

της οθόνης που θα χρησιμοποιήσει ο χρήστης καθώς και η μη γνώση του μεγέθους της Βάσης Δεδομένων, ένα αντικείμενο μπορεί να έχει αναφορές σε όλα τα υπόλοιπα αντικείμενα που είναι αποθηκευμένα στη Βάση Δεδομένων. Αυτό κανονικά δεν αποτελεί πρόβλημα καθώς αναπαρίσταται με ένα πεδίο που συνήθως είναι το κύριο κλειδί ενός άλλου αντικειμένου, ωστόσο στις Object Oriented Βάσεις Δεδομένων όπου γίνεται αναπαράσταση ολόκληρου του αντικειμένου, όλα τα πεδία αυτού, αποτελεί σχεδιαστικό δίλλημα. Η λύση σε αυτό το πρόβλημα δόθηκε μέσω του View τύπου Λίστα που προσφέρει το Λειτουργικό Σύστημα Android, έτσι ανεξάρτητα από το μέγεθος της οθόνης, την έκδοση του Λειτουργικού Συστήματος καθώς και το μέγεθος της Βάσης Δεδομένων ο χρήστης μπορεί να θέσει περιορισμούς και να προβάλλει τα δεδομένα που πληρούν τους περιορισμούς αυτούς. Το επόμενο εμπόδιο που μας παρουσιάστηκε ήταν στο πως θα χτιστεί το ερώτημα για να εκτελεστεί στη συνέχεια. Επειδή τους περιορισμούς τους αποθηκεύαμε προσωρινά σε μία μεταβλητή τύπου λίστα έπρεπε να δημιουργήσουμε μία μέθοδο η οποία θα έχτιζε το ερώτημα ανεξαρτήτως βάθους, στην περίπτωση που στη Βάση Δεδομένων υπάρχουν πολλά αποθηκευμένα αντικείμενα και όλα έχουν κάποια αναφορά σε κάποιο άλλο θα έπρεπε ο χρήστης να μπορεί να ξεκινήσει από ένα αντικείμενο, να προσπελάσει όλα τα υπόλοιπα και να θέσει περιορισμό μόνο στο τελευταίο αντικείμενο. Η λύση ήταν η δημιουργία μίας μεθόδου που με τη χρήση της αναδρομής πετυχαίνει την επίλυση αυτού του προβλήματος χτίζοντας τμηματικά το ερώτημα σε κάθε επανάληψη. Τελευταίο εμπόδιο το οποίο δυστυχώς δεν μπορέσαμε να προβούμε στην επίλυσή του ήταν η μεταφορά από τις Σχεσιακές Βάσεις Δεδομένων της δυνατότητας ο χρήστης να μπορεί να αποφασίσει για κάθε περιορισμό ξεχωριστά αν θέλει να συνδέεται με τον επόμενο με τη χρήση λογικού “και” ή με τη χρήση του λογικού “ή”. Έπειτα από διεξοδική έρευνα τόσο στην τεκμηρίωση που συνοδεύει την τεχνολογία db4o, documentation, όσο και διαφόρων τεχνικών και μη επιστημονικών μελετών, καθώς και τη διενέργεια πειραμάτων με τη χρήση κώδικα δεν καταφέραμε να ενώσουμε τον κάθε περιορισμό ξεχωριστά με έναν διαφορετικό λογικό σύνδεσμο, καταλήξαμε στο συμπέρασμα ότι με τις βιβλιοθήκες, δυνατότητες που προσφέρει έως την ημερομηνία συγγραφής αυτής της πτυχιακής η Βάση Δεδομένων db4o δεν είναι δυνατό κάτι τέτοιο. Ευελπιστούμε πως στο προσεχές μέλλον μέσω της διαρκούς ανάπτυξης της db4o θα υπάρξει λύση σε αυτό το πρόβλημα.

Πολλαπλά είναι τα χαρακτηριστικά τα οποία θα μπορούσαν να εμπλουτίσουν την εφαρμογή μας. Μερικά από αυτά είναι η υποστήριξη προβολής και τοπικά αποθηκευμένης Βάσης Δεδομένων τύπου db4o στη φορητή συσκευή έτσι ώστε ο χρήστης να μην χρειάζεται να συνδεθεί σε απομακρυσμένη Βάση Δεδομένων με ότι αυτό συνεπάγεται από το αντίκτυπο του δικτύου στην αποτελεσματικότητα της εφαρμογής αλλά και στην εμπειρία χρήστη. Έπειτα θα μπορούσαμε να προσθέσουμε την αποθήκευση των περισσοτέρων στοιχείων σύνδεσης του χρήστη, η αποθήκευση του κωδικού πρόσβασης μπορεί να προκαλέσει κενό ασφάλειας με μη επιθυμητά αποτελέσματα. Στη συνέχεια πολύ σημαντικό χαρακτηριστικό θα ήταν η προβολή των αντικειμένων τύπου Collection, πράγμα που είναι δύσκολο καθώς ο χρήστης μπορεί να θέλει να προβάλλει μία λίστα που

περιέχει στοιχεία διαφορετικών τύπων δεδομένων. Τέλος, ενα χαρακτηριστικό το οποίο θα μπορούσε να συμβάλλει στην εμπειρία του χρήστη είναι η χρήση πολύπλοκων χειρονομιών (Gestures), ωστόσο αυτό θα μπορούσε να χρησιμοποιηθεί μόνο για την επιλογή προηγούμενης σύνδεσης καθώς και τη μετάβαση από μία Activity σε άλλη.

ΑΝΑΦΟΡΕΣ

[1] https://www.android.com/history/#/marshmallow

[2] http://www.odbms.org/wp-content/uploads/2013/11/db4o-7.10-tutorial-net.pdf

[3] http://nosql-database.org/

[4] https://developer.android.com/studio/index.html

[5] https://www.jetbrains.com/idea/

[6] http://www.openhandsetalliance.com/android_overview.html

[7] https://en.wikipedia.org/wiki/Android_(operating_system)

[8] https://design.google.com/

[9] http://www.datastax.com/nosql-databases

[10] http://planetcassandra.org/what-is-nosql/

[11] http://supportservices.actian.com/versant/default.html

[12] https://en.wikipedia.org/wiki/Db4o

[13] http://www.gamlor.info/wordpress/2009/10/db4o-queries-in-java-or-queries-without-linq/

[14] https://www.oracle.com/java/index.html

[15] http://www.tutorialspoint.com/java/java_collections.htm

[16] https://developer.android.com/reference/android/content/Context.html

[17] https://developer.android.com/training/basics/data-storage/shared-preferences.html

[18] Eugen P., Jackson JSON Views, Jackson.Spring (Ebook)

[19]https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html

[20] https://developer.android.com/reference/android/R.attr.html#listDivider

[21] Jackson W. (2014), Android Apps for Absolute Beginners Third Edition, Apress Media LLC, New York City, United States of America

[22] https://developer.android.com/studio/profile/dev-options-overdraw.html

ΒΙΒΛΙΟΓΡΑΦΙΑ

Dutson, P. (2016), Android Development Patterns Best Practices for Professional Developers, Pearson Education, United States of America

Eugen, P., Jackson JSON Views, Jackson.Spring (Ebook)

Felkern, D. and Dobbs, J. (2011), Android Application for Dummies, Wiley Publishing Inc., Indianapolis, United States

Gargenta, M. (2011), Learning Android, O'Reilly Media Inc., Sebastopol, United States of America

Hodson, R. Foreword by Jebaraj, D. (2014), Android Programming Succinctly, Syncfusion Inc., Morrisville, United States of America

Jackson, W. (2014), Android Apps for Absolute Beginners Third Edition, Apress Media LLC, New York City, United States of America

Kuniawan, B. (2015), Android Application Development: A Beginner's Tutorial, BrainySoftware, Montreal, Canada

Kuniawan, B. (2015), Java for Android Second Edition, BrainySoftware, Montreal, Canada

Murphy, M. L. (2011), The Busy Coder's Guide to Android Development, Version 3.6, CommonsWare LLC, United States of America

Sillars, D. (2015), High Perfomance Android Apps Improve Ratings with Speed, Optimazations and Testing, O'Reilly Media Inc., Sebastopol, United States of America

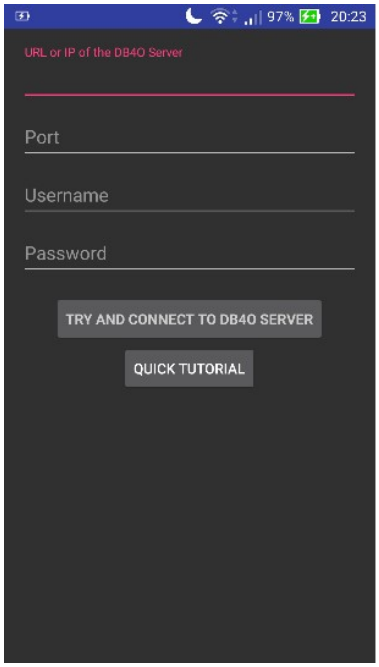
Zigurd, M., Laird, D., G. Blake, M.,and Masumi, N. (2011), Programming Android, O'Reilly Media Inc., Sebastopol, United States of America

ΟΔΗΓΟΣ ΧΡΗΣΗΣ ΛΟΓΙΣΜΙΚΟΥ

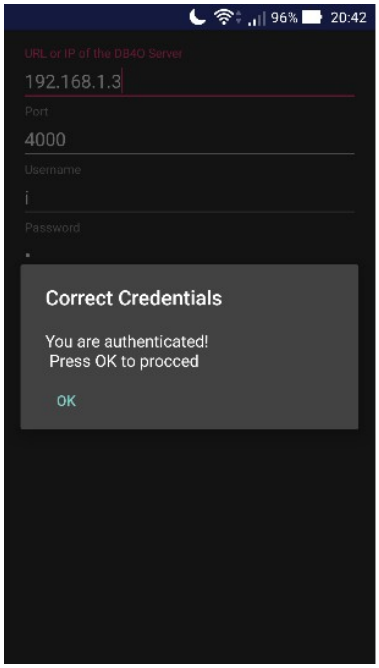
DB4O Viewer

Quick Tutorial

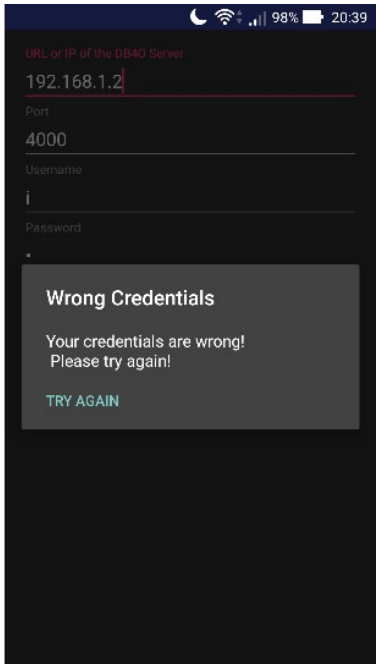
Login Activity



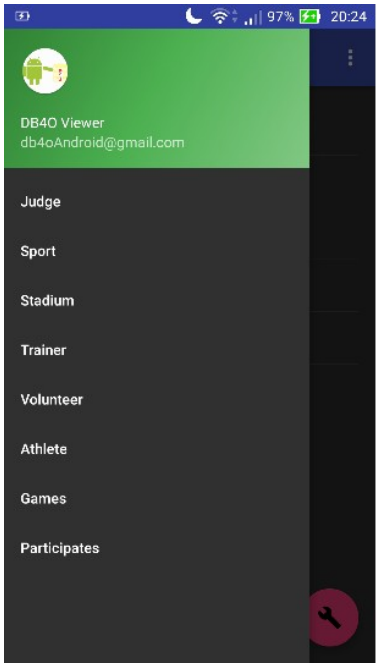
In this Activity the User must fill the form with the desired credentials and hit the “TRY AND CONNECT TO DB4O SERVER” button. If the User hits the “Quick Tutorial” button this tutorial will be opened at his prefered Web Browser.



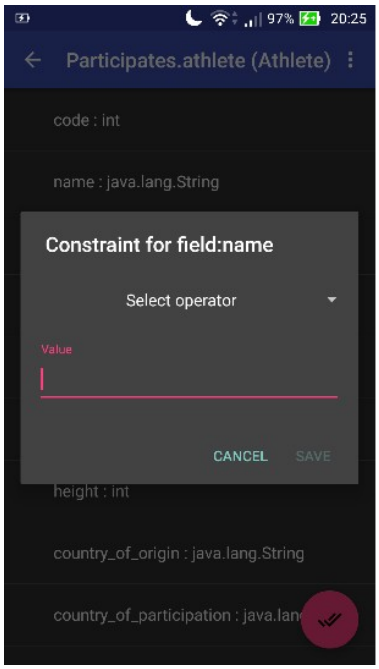
Whether the Credentials were correct or not, the appropriate mesage will be displayed.



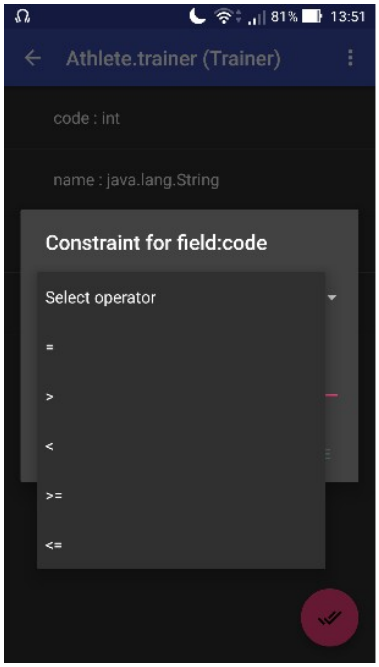
Object Selection Activity



When the User hits the app icon button on the top level of the screen the navigator drawer window will be revealed. There are all the Objects stored by the User on the db4o Database.

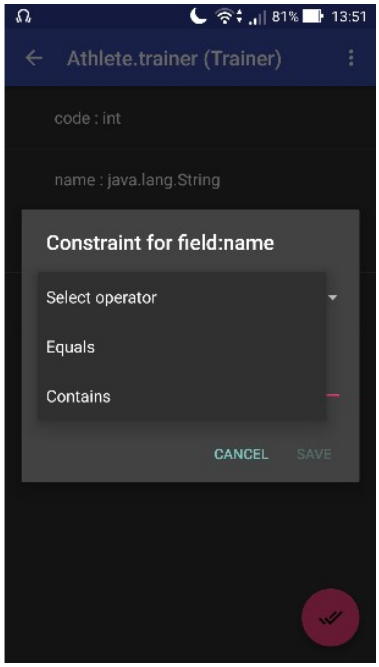


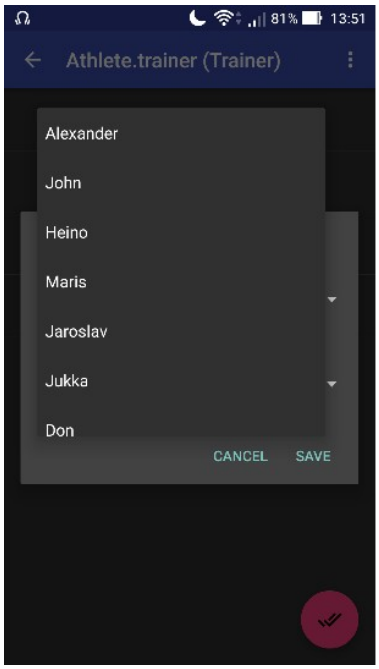
In every Constraint Dialog there are the Selector spinner, where the User can choose the Operator to be used, the Value Field which can be an EditText or a Spinner. Finally, the User can abort by hitting the “Cancel” button or save the constraint by hitting the “Save” button.



If the type of te selected field type is numeric or date then the User can select from the Selector Spinner the Operators greater,equals, etc.

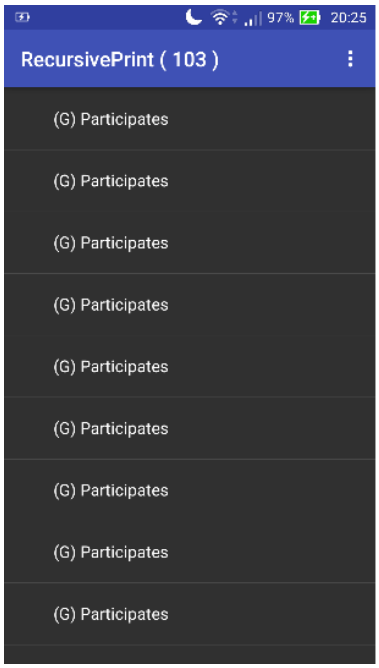
If the selected field type is String then the User can select the “Equals” or “Contains” operators.





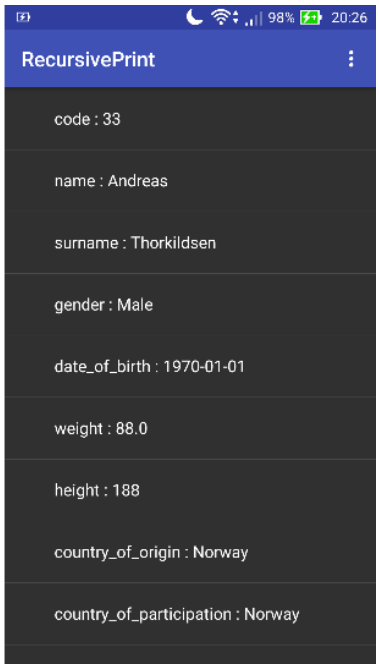
In the “Equals” case, the User will select the desired value from the Spinner Value.

Show Results Activity



Firstly, the User has to select one of the Objects that match his constraints.

After that the User can see the values of all the fields.



Preview Selected Constraints

This Activity can be launched from both the Constraints and the Results Activity.

The User can choose a constraint from the drawer and thew preview it on the WebView.