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

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Διαχείριση του Προγράμματος μαθημάτων του Τμήματος Πληροφορικής με χρήση Αντικειμενοστρεφών Βάσεων Δεδομένων

Του φοιτητή

Θεοφίλου Θεόφιλος

Αρ. Μητρώου: 02/1993

Επιβλέπων καθηγητής

Κεραμόπουλος Ευκλείδης

Θεσσαλονίκη 2014

ΠΡΟΛΟΓΟΣ

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία στοχεύει στη δημιουργία μίας client-server εφαρμογής για τη διαχείριση του προγράμματος μαθημάτων ενός Τμήματος Ανώτατου Εκπαιδευτικού Ιδρύματος και συγκεκριμένα του Τμήματος Μηχανικών Πληροφορικής του ΑΤΕΙΘ.

Η υλοποίηση αυτής της client εφαρμογής έγινε με τη βοήθεια της αντικειμενοστραφούς γλώσσας προγραμματισμού, Java, η υλοποίηση της διεπιφάνειας με τη βοήθεια της Java Swing, ενώ για τις ανάγκες της Βάσης Δεδομένων χρησιμοποιήθηκε η Αντικειμενοστραφής Βάση Δεδομένων db4o της Versant. Η εν λόγω εφαρμογή κατασκευάστηκε με το λογισμικό εργαλείο IDE NetBeans.

Στα κεφάλαια που ακολουθούν περιγράφονται αναλυτικά οι τεχνολογίες που χρησιμοποιήθηκαν και παρατίθεται ένας οδηγός χρήσης της δημιουργηθείσας εφαρμογής

ABSTRACT

This graduation project aims at creating a client-server application for managing the courses' schedule of one University Department and specifically the Department of Informatics of ATEITH.

The implementation of this client application was done with the support of the object-oriented programming language, Java, the implementation of the interface was done with the support of Java Swing, and for the needs of the database it was used the object-oriented database db4o by Versant. This application was built with the software tool IDE NetBeans.

The following chapters detail the technologies that were used and is cited a user guide for the created application.

ΠΕΡΙΕΧΟΜΕΝΑ

Ευρετήριο σχημάτων

ΕΙΣΑΓΩΓΗ

Περιγραφή του προβλήματος

Η δόμηση του προγράμματος μαθημάτων για ένα τμήμα τριτοβάθμιας εκπαίδευσης αποτελεί ένα σύνθετο πρόβλημα. Στην παρούσα εργασία το πρόβλημα συγκεκριμενοποιείται για το Τμήμα Μηχανικών Πληροφορικής ΑΤΕΙΘ. Στο εν λόγω τμήμα διδάσκονται κατά το χειμερινό εξάμηνο τα μαθήματα των μονών εξαμήνων και κατά το εαρινό των ζυγών. Επίσης, κάθε τυπικό εξάμηνο περιλαμβάνει 5 μαθήματα, με αποτέλεσμα κατά το χειμερινό εξάμηνο να πρέπει να διδαχθούν 20 μαθήματα και κατά το εαρινό 15. Ακόμα, υπάρχουν και κάποια μαθήματα επιλογής που διδάσκονται σε κάθε εξάμηνο. Αυτό σημαίνει ότι προκύπτει ένας μέσος όρος 20 μαθημάτων ανά εξάμηνο. Επιπλέον, κάθε μάθημα αποτελείται κατά μέσο όρο από 4 ώρες θεωρητικών διαλέξεων και τις περισσότερες φορές και από 2 ώρες εργαστηριακών. Αξίζει επίσης να σημειωθεί ότι για κάθε εργαστηριακό μάθημα δημιουργούνται συνήθως 3-4 τμήματα, μερικές φορές μπορεί να φτάσουν και τα 7. Κάνοντας έναν πρόχειρο υπολογισμό, οι ώρες που θα χρειάζεται να κατανεμηθούν είναι περίπου 250! Όπως γίνεται εύκολα αντιληπτό κάτι τέτοιο είναι εξαιρετικά δύσκολο και για αυτό το λόγο απαιτείται η εύρεση λύσης στο εν λόγω πρόβλημα.

Το άτομο που αναλαμβάνει τη δημιουργία του προγράμματος μαθημάτων, στην πιο απλή περίπτωση, λαμβάνει από το Τμήμα τα δεδομένα (ποια μαθήματα θα διδαχθούν, πόσες θεωρητικές και εργαστηριακές ώρες θα έχουν, πόσα εργαστηριακά τμήματα θα δημιουργηθούν, κλπ.) Αν οι απαιτήσεις ως προς τον προγραμματιστή σταματούσαν εδώ, ίσως να μπορούσε να επιλυθεί το πρόβλημα απλώς φτιάχνοντας ένα πίνακα κατανομής αιθουσών και ωρών. Ωστόσο, στο συγκεκριμένο πρόβλημα υπάρχει πληθώρα απαιτήσεων, και μάλιστα είναι ετερογενείς μεταξύ τους. Συγκεκριμένα, υπάρχουν απαιτήσεις που προέρχονται από το ίδιο το Ίδρυμα ή/και το Τμήμα, όπως για παράδειγμα η διάθεση αιθουσών για άλλους σκοπούς (συνελεύσεις, Μεταπτυχιακές σπουδές, διαλέξεις άλλων τμημάτων, κ.ά.). Ανάλογες απαιτήσεις οφείλονται και στους διδάσκοντες. Καθένας πρέπει να διδάξει ένα συγκεκριμένο αριθμό ωρών που καθορίζεται ανάλογα με τη σχέση (π.χ. εργασιακή, εκπαιδευτική, διοικητική) που έχει με το Τμήμα ή/και το Ίδρυμα. Επιπλέον, απαιτήσεις προκύπτουν ακόμα και από το πρόγραμμα σπουδών. Για παράδειγμα, τα μαθήματα που βρίσκονται στο ίδιο τυπικό εξάμηνο δεν μπορούν να αλληλεπικαλύπτονται χρονικά. Οι παραπάνω απαιτήσεις αναφέρθηκαν ενδεικτικά για την απόκτηση μιας πρώτης αντίληψης του μεγέθους του προβλήματος.

Καθώς η ανάλυση των απαιτήσεων του εν λόγω προβλήματος μπορεί να καλύψει ολόκληρο κεφάλαιο, δίνεται περισσότερη προσοχή στην αναγκαιότητα ύπαρξης μίας εφαρμογής που να διαχειρίζεται τους διδάσκοντες, τα τμήματα, τις

αίθουσες, έχοντας ως γνώμονα κάθε φορά τις αντίστοιχες απαιτήσεις. Επίσης με τη σειρά του ο χρήστης του προγράμματος να μπορεί εύκολα, γνωρίζοντας ποιες απαιτήσεις δεν τηρούνται, να πραγματοποιεί τις αναγκαίες αλλαγές.

Ο Αντικειμενοστρεφής Προγραμματισμός

Η παρούσα εργασία δεν αποσκοπεί απλά στην επίλυση του παραπάνω προβλήματος. Σκοπός, επίσης, είναι και η παρουσίαση της σημασίας και της ευκολίας της χρήσης μιας αντικειμενοστρεφούς βάσης δεδομένων. Και για την επίτευξη αυτού χρειάζεται ολόκληρη η διαδικασία επίλυσης του προβλήματος να βασίζεται στον αντικειμενοστρεφή προγραμματισμό και όχι απλώς στη χρήση του.

Στην ανάπτυξη λογισμικού αυτό που επιβάλει αλλαγές σε ένα πολύ μεγάλο ποσοστό είναι το στάδιο της «Εξαγωγής Και Ανάλυσης Απαιτήσεων». Αυτό το στάδιο είναι τόσο σημαντικό που, όχι και πολύ παλιά, αν δεν ολοκληρωνόταν επιτυχώς, δεν μπορούσε να γραφτεί γραμμή κώδικα. Σήμερα υπάρχουν τεχνικές που επιτρέπουν σε αυτό το κομμάτι να γίνει παράλληλα με τη συγγραφή κώδικα. Ωστόσο, αν τροποποιηθούν οι απαιτήσεις κατά την ανάπτυξη λογισμικού, είναι πιθανό μεγάλα κομμάτια κώδικα να αλλάξουν ή ακόμα και να μη χρησιμοποιηθούν.

Παρ’ όλες τις δυσκολίες που προσθέτουν οι απαιτήσεις λογισμικού, χρήσιμο θα ήταν να εξετασθεί λίγο και το τεχνικό κομμάτι. Ο άνθρωπος και ο υπολογιστής επικοινωνούν σε δύο εντελώς διαφορετικές γλώσσες. Και η γλώσσα είναι αυτή που καθορίζει το πώς αντιλαμβανόμαστε τον κόσμο και τα προβλήματα. Στην καθημερινότητά του ο άνθρωπος λειτουργεί (τουλάχιστον υποσυνείδητα) με αντικείμενα, ξεχωρίζει οντότητες. Κάθε πρόβλημα το αντιλαμβάνεται σαν ένα σύνολο αντικειμένων που πρέπει να εκτελέσουν κάποιες ενέργειες και να αλληλεπιδράσουν. Εν αντιθέσει, ο υπολογιστής «αντιλαμβάνεται» στο δυαδικό σύστημα (τα λεγόμενα 0 και 1).

Έχουν καταβληθεί μεγάλες προσπάθειες ώστε να υπάρξει μια πλατφόρμα προγραμματισμού που θα κάνει τον υπολογιστή να έχει μια αντίληψη σαν του ανθρώπου. Οι γενεές στις γλώσσες προγραμματισμού έχουν να κάνουν με τα βήματα που γίνονται προς σ’ αυτό το τομέα. Στην πρώτη γενεά βρίσκεται η γλώσσα μηχανής, όπου οι εντολές ήταν καθαρά στο δυαδικό σύστημα. Στη δεύτερη δημιουργήθηκαν οι συμβολογλώσσες όπου λέξεις – εντολές μεταφράζονται άμεσα σε γλώσσα μηχανής. Στην τρίτη σταμάτησε η αντιστοιχία «ένα – προς- ένα» των λέξεων – εντολών με τις εντολές στην γλώσσα μηχανής. Οι γλώσσες προγραμματισμού τρίτης γενεάς δίνουν την δυνατότητα στον προγραμματιστή να δίνει τις εντολές «πιο κοντά» στη δική του γλώσσα. Και σ’ αυτή τη γενεά ουσιαστικά ξεκινά η έννοια της αντίληψης του υπολογιστή όπως προαναφέρθηκε. Στην τέταρτη γενεά οι γλώσσες δε χρησιμοποιούνται για να κωδικοποιήσουν έναν αλγόριθμο, αλλά περιγράφουν το πρόβλημα και τις

απαιτήσεις του. Οι γλώσσες τέταρτης γενεάς δεν έχουν καταφέρει μέχρι σήμερα να είναι κατάλληλες για την επίλυση της πλειοψηφίας των προβλημάτων. Γι' αυτό και χρησιμοποιούνται ακόμα γλώσσες τρίτης γενεάς με, ανάλογα την περίπτωση, βοηθητική χρήση μιας γλώσσας τέταρτης γενεάς.

Ο αντικειμενοστρεφής προγραμματισμός είναι μια επέκταση για τις γλώσσες τρίτης γενεάς. Δίνει την δυνατότητα του απευθείας ορισμού οντοτήτων (με τη δημιουργία κλάσεων) των οποίων τα στιγμιότυπα (αντικείμενα) είναι αυτά που αλληλεπιδρούν μεταξύ τους στην εκτέλεση της εφαρμογής. Τα αντικείμενα έχουν δύο στοιχεία, ένα κοινό για όλα τα αντικείμενα της ίδιας κλάσης και ένα που τα ξεχωρίζει. Το κοινό στοιχείο είναι οι ενέργειες (μέθοδοι) που μπορεί να κάνει και μέσω αυτών τα αντικείμενα αλληλεπιδρούν. Ο διαχωρισμός των αντικειμένων γίνεται μέσω των χαρακτηριστικών τους, που είναι οι μεταβλητές.

Τώρα πια, από την στιγμή της ανάληψης επίλυσης ενός προβλήματος, μπορεί να μεταφερθεί άμεσα η ανθρώπινη αντίληψη για το πρόβλημα σε κώδικα. Αυτό βοηθάει τόσο στον άμεσο έλεγχο της ορθής ανθρώπινης αντίληψης, όσο και στη βελτίωσή της.

Αν και η έννοια του αντικειμενοστρεφούς προγραμματισμού υπάρχει εδώ και δεκαετίες, άρχισε να χρησιμοποιείται ευρέως τη δεκαετία του '90. Χρειάστηκαν μερικά χρόνια ακόμα για ν' αναπτυχθούν οι αντικειμενοστρεφείς γλώσσες προγραμματισμού, αλλά και για να αρχίσουν να χρησιμοποιούνται τεχνικές ανάπτυξης που να βασίζονται στην αντικειμενοστρέφεια.

Βάσεις Δεδομένων

Το θέμα της αποθήκευσης και ανάκτησης των δεδομένων απασχολούσε πάντα την Επιστήμη της Πληροφορικής. Η μονιμότητα (persistency) των δεδομένων μιας εφαρμογής έδωσε μεγάλη ώθηση στη χρήση των ηλεκτρονικών υπολογιστών, μιας και ανοίχτηκαν πολλά νέα πεδία εφαρμογών. Για πολλά χρόνια η υλοποίηση της αποθήκευσης και ανάκτησης δεδομένων ήταν αποκλειστικά ευθύνη του προγραμματιστή. Έπρεπε να προσθέτει στις εφαρμογές του τον κατάλληλο κώδικα για τον χειρισμό των αποθηκευτικών μέσων, μέχρι και για την απόδοση των δεδομένων στις κατάλληλες μεταβλητές. Επειδή, όμως, όλο αυτό αποτελεί μια δύσκολη διαδικασία χρειάστηκε να βρεθούν τρόποι αυτοματοποίησής της.

Η Βάση Δεδομένων είναι μια δομή αυτοματοποίησης της διαδικασίας μονιμοποίησης των δεδομένων. Ουσιαστικά, πρόκειται για εφαρμογές που αναλαμβάνουν την αποθήκευση και ανάκτηση των δεδομένων στο αποθηκευτικό μέσο και όλη αυτή τη διαδικασία την παρουσιάζουν με ένα ιδεατό τρόπο στον προγραμματιστή. Το μόνο που έχει να κάνει ο προγραμματιστής είναι να επιλέγει και να αναθέτει τα δεδομένα με τον κατάλληλο τρόπο. Κριτήριο για την επιλογή της βάσης δεδομένων είναι το πώς παρουσιάζει τα δεδομένα στον προγραμματιστή.

Οι λεπτομέρειες της φυσικής υλοποίησης της όλης διαδικασίας μας αφορούν ιδιαίτερα, μόνο όταν οι εφαρμογές μας έχουν συγκεκριμένες απαιτήσεις σε αποδόσεις.

Κυρίαρχη προτίμηση εδώ και μερικές δεκαετίες στις βάσεις δεδομένων αποτελούν οι σχεσιακές. Οι σχεσιακές βάσεις δεδομένων παρουσιάζουν τα δεδομένα σε μορφή δυσδιάστατων πινάκων. Κάθε πίνακας αντιπροσωπεύει μια οντότητα, με την κάθε στήλη να αντιπροσωπεύει ένα χαρακτηριστικό της και κάθε γραμμή ένα στιγμιότυπό της. Αυτοί οι πίνακες δύναται να έχουν και κάποια συσχέτιση μεταξύ τους.

Οι σχεσιακές βάσεις δεδομένων συνοδεύονται και από μια γλώσσα προγραμματισμού, την Structured Query Language (SQL). Αυτή η γλώσσα βοηθά στην πρόσβαση στα δεδομένα που αποθηκεύονται στην βάση. Το μεγάλο πλεονέκτημα που παρουσιάζεται είναι πως όποια γλώσσα προγραμματισμού και αν επιλεχθεί για τη δημιουργία μιας εφαρμογής, ο κώδικας SQL παραμένει ουσιαστικά ίδιος. Επίσης, μεγάλο προτέρημα αποτελεί η λογική της. Ο κώδικας δεν είναι εντολές που ορίζουν το πώς θα ανασυρθούν τα δεδομένα, αλλά το ποια είναι τα ζητούμενα δεδομένα.

Φτάνοντας στο σήμερα που οι σχεσιακές βάσεις δεδομένων έχουν αναπτυχθεί τόσο που μπορούν να χρησιμοποιηθούν άμεσα από οποιαδήποτε εφαρμογή, από την πιο μικρή και απλή, μέχρι και σε εφαρμογές που χρησιμοποιούνται από μεγάλες κυβερνήσεις και πολυεθνικές εταιρείες. Επιπλέον, προσφέρουν πολλές λύσεις και ευκολίες σε κάθε χρήστη.

Οπότε εδώ τίθεται το ερώτημα, γιατί δεν προτιμήθηκε μια σχεσιακή βάση δεδομένων για την επίλυση του προβλήματος της παρούσας εργασίας. Ο λόγος είναι ότι μια σχεσιακή βάση δεδομένων είναι ένα σύνολο από πίνακες. Η ζητούμενη εφαρμογή όμως, όπως προαναφέρθηκε, αναπτύσσεται με αντικειμενοστρεφή γλώσσα προγραμματισμού, με πλήρη αντικειμενοστρεφή λογική. Αυτό σημαίνει ότι το μοντέλο οντοτήτων θα πρέπει να υποστεί κάποιες τροποποιήσεις όταν θα εφαρμοστεί στη βάση. Επιπλέον, η σχεσιακή βάση δεδομένων δεν αντιλαμβάνεται τις κλάσεις, τα αντικείμενα, τις μεταβλητές όπως γίνονται αντιληπτές μέσα σε μια αντικειμενοστρεφή γλώσσα. Είναι απλά γραμμές ή στήλες σε έναν πίνακα.

Παρότι υπάρχουν εργαλεία που βοηθούν να γεφυρωθεί το χάσμα, και αρκετά αποδοτικά, η ουσία είναι ότι ο προγραμματιστής έχει ακόμα μία υποχρέωση, χρειάζεται να παρακολουθεί αυτόν τον «μετατροπέα». Όταν θα χρειαστεί να κάνει αλλαγές στο μοντέλο οντοτήτων της εφαρμογής πρέπει να τις κάνει και στον «μετατροπέα», αλλιώς θα υπάρχει κίνδυνος να αποθηκεύει και να ανακτά δεδομένα, χωρίς αντίκρισμα. Επιπλέον, δεν είναι και πολύ δύσκολη η παραγωγή κάποιου λάθους ή διπλότυπου στην βάση (γνωστό και ως «πλεονάζουσα πληροφορία»). Πέρα από την κατανάλωση του χώρου αποθήκευσης, αυτό μπορεί να οδηγήσει σε ασυνέπεια των δεδομένων.

Οι αντικειμενοστρεφής βάσεις δεδομένων δίνουν στον προγραμματιστή την ίδια όψη στα δεδομένα που έχει και μέσα στην εφαρμογή του, δηλαδή αυτή των αντικειμένων. Πέρα από την λογική έννοια, με την έννοια αντικείμενο νοείται ένας συγκεκριμένος χώρος της μνήμης που περιέχει κάποια δεδομένα. Όταν αποθηκεύεται το αντικείμενο, μια αντικειμενοστρεφής βάση δεδομένων διαβάζει αυτή την περιοχή της μνήμης και την αποθηκεύει. Στην ανάκτηση, η βάση δεσμεύει μια αντίστοιχη περιοχή στην μνήμη και την επαναδομεί όπως ήταν προηγουμένως.

Αποτέλεσμα όλων των προηγουμένων είναι ότι δεν χρειάζεται κάποια επιπλέον διαδικασία για την αποθήκευση και την ανάκτηση των δεδομένων. Ο προγραμματιστής ακολουθεί μέχρι τέλους το μοντέλο οντοτήτων και πραγματοποιώντας όποιες τροποποιήσεις σ' αυτό γίνονται άμεσα εφαρμόσιμες και στη βάση δεδομένων. Επιπλέον, όλες οι λειτουργίες στη βάση δεδομένων γίνονται με τον κώδικα της γλώσσας προγραμματισμού που χρησιμοποιείται κάθε φορά, προσφέροντας τη δυνατότητα για έλεγχο και αποσφαλμάτωση κατά την ώρα της κωδικοποίησης. Στις σχεσιακές βάσεις αυτό γίνεται με την εκτέλεση της εφαρμογής ή με εξωτερικά εργαλεία.

Η γλώσσα προγραμματισμού Java

Η Sun Microsystems τη δεκαετία του '90 δημιούργησε τη γλώσσα προγραμματισμού Java, εμπνευσμένη κυρίως από την C++. Αυτό που οδήγησε στη δημιουργία της και μετέπειτα στη μεγάλη διάδοσή της, είναι ότι οι εφαρμογές της μπορούν άμεσα να εκτελεστούν σε πολλά λειτουργικά συστήματα. Μέχρι τότε όλες οι γλώσσες προγραμματισμού χρειαζόντουσαν ξεχωριστές βιβλιοθήκες για κάθε λειτουργικό σύστημα και πολλές εξ' αυτών δεν τα υποστήριζαν όλα. Ακόμα και αυτές που είχαν τη δυνατότητα, χρειαζόταν ειδική μέριμνα στον κώδικα και ξεχωριστή μεταγλώττιση για την κάθε πλατφόρμα. Η Sun δημιούργησε κάτι σαν ένα «μίνι λειτουργικό» που έτρεχε μέσα στο ήδη υπάρχον και επέτρεπε στην Java εφαρμογή, χωρίς επιπλέον εργασία από τον προγραμματιστή, να εκτελείται. Αυτό το «μίνι λειτουργικό» – ή αλλιώς framework – αποτέλεσε στην αρχή πεδίο αντιπαράθεσης, αλλά αργότερα ακόμα και η Microsoft το καθιέρωσε στις δικές της γλώσσες. Τα πρώτα χρόνια κύριο πεδίο εφαρμογής της Java ήταν οι εφαρμογές για ιστοσελίδες. Λόγω της δυνατότητας της Java ήταν πολύ εύκολο να δημιουργηθούν εφαρμογές – τα λεγόμενα applet– που θα εκτελούνται μέσω του περιηγητή.

Μιας και η εφαρμογή θα ήταν χρήσιμο να μπορεί να εκτελεστεί σε διαφορετικές πλατφόρμες, η χρήση της Java ήταν σχεδόν μονόδρομος για την παρούσα εργασία. Καταλυτικό ρόλο έπαιξε και το γεγονός ότι είναι μια γλώσσα που χρησιμοποιείται κατά κόρον στο πρόγραμμα σπουδών του Τμήματος Πληροφορικής ΑΤΕΙΘ. Η ήδη υπάρχουσα εμπειρία στη Java, θα βοηθούσε στην περισσότερη αφοσίωση στο κύριο θέμα της εργασίας, που είναι η καθαρά

αντικειμενοστρεφής ανάπτυξη μιας εφαρμογής με την χρήση αντικειμενοστρεφούς βάσης.

Για την ανάπτυξη της διεπαφής με τον χρήστη, χρησιμοποιήσαμε το Swing Framework της Java. Η Swing αποτελεί την κύρια βιβλιοθήκη της Java για τον προγραμματισμό GUIs (Graphical User Interfaces) και γενικότερα χρησιμοποιείται από τους προγραμματιστές για τη σύνδεση της Java με το γραφικό περιβάλλον. Αναπτύχθηκε με σκοπό τη βελτίωση του AWT (Abstract Window Toolkit) το οποίο ήταν το πρώτο πακέτο της Java για GUI, αν και όχι ιδιαίτερα επιτυχημένο.

Αξίζει να σημειωθεί ότι στη Swing υποστηρίζονται όλα τα συστατικά (components) που μπορεί να συναντήσει κάποιος σε ένα σύγχρονο περιβάλλον διεπαφής χρήστη, από τα πολύ γνωστά, όπως παράθυρα, κουμπιά, μενού, ετικέτες, κλπ., μέχρι και πιο προηγμένα συστατικά όπως καρτέλες, παράθυρα κύλισης, δέντρα, πίνακες και λίστες.

Βασική λειτουργία της Swing είναι ότι τα συστατικά (components) που προαναφέρθηκαν πυροδοτούν συμβάντα (events). Τα αντικείμενα-ακροατές (Listeners) με τη σειρά τους χειρίζονται αυτά τα συμβάντα, χρησιμοποιώντας ειδικές μεθόδους. Για παράδειγμα, ένα κουμπί πυροδοτεί ένα ActionEvent, όποτε ο χρήστης το πατάει και πραγματοποιούνται οι κατά περίπτωση επιθυμητές ενέργειες καλώντας τις κατάλληλες μεθόδους του listener.

Το περιβάλλον ανάπτυξης (integrated development environment, IDE) που χρησιμοποιήθηκε ήταν το Netbeans. Η επιλογή του συγκεκριμένου λογισμικού έγινε και πάλι με γνώμονα την εμπειρία αλλά και λόγω της ευρείας χρήσης του από τους προγραμματιστές Java. Επίσης, έτσι γίνεται φανερό ότι η χρήση μιας αντικειμενοστρεφούς βάσης είναι απρόσκοπτη, δεν προσθέτει «βάρος» ή ιδιαίτερα επιπλέον φόρτο εργασίας.

Επιπρόσθετα, σημαντικό ρόλο στην επιλογή αυτής της γλώσσας και του αντίστοιχου εργαλείου λογισμικού κατέχει και το γεγονός ότι και τα δύο διατίθενται δωρεάν για ακαδημαϊκή χρήση.

Η αντικειμενοστρεφής βάση δεδομένων db4o

Η db4o είναι μία ανοιχτού κώδικα, υψηλής απόδοσης αντικειμενοστρεφής βάση δεδομένων και προορίζεται για υποστηρικτική βοήθεια των Java και .NET προγραμματιστών. Επιτρέπει στους εν λόγω προγραμματιστές να αποθηκεύουν και να ανακτούν εύκολα αντικείμενα από το σύστημα.

Η αντικειμενοστρεφής βάση δεδομένων db4o αποτελεί μια φρέσκια τεχνολογία στις βάσεις δεδομένων. Συγκεκριμένα, αναπτύχθηκε από τον επικεφαλή αρχιτέκτονα λογισμικού Carl Rosenberger το 2000. Ξεκίνησε να χρησιμοποιείται επιτυχώς από επιχειρήσεις και για ακαδημαϊκούς σκοπούς πριν την επίσημη έναρξη της εμπορικής της λειτουργίας το 2004 από την τότε

νεοσυσταθείσα ιδιωτική εταιρεία Db4objects Inc. Το 2008 η db4o OODBMS εξαγοράστηκε από την εταιρεία Versant και ανήκει σε αυτή μέχρι σήμερα.

Η ανάπτυξη της db4o έχει επεκταθεί σε 170 διαφορετικές χώρες και χρησιμοποιείται ευρέως από εταιρείες κολοσσούς στον κλάδο τους, όπως Boeing, Bosch, Intel, Ricoh και Seagate.

Η db4o αποτελεί μία επιτυχημένη αντικειμενοστρεφή βάση δεδομένων, καθώς μπορεί να διαχειριστεί ενδογενή (native) αντικείμενα. Συγκριτικά με τις Σχεσιακές Βάσεις, απαιτεί πολύ λιγότερο κώδικα, ενώ συγκριτικά με άλλες αντικειμενοστρεφείς βάσεις, προσφέρει υψηλότερη απόδοση.

Επιπρόσθετα η db4o παρουσιάζει πληθώρα πλεονεκτημάτων, όπως ότι υποστηρίζεται από όλα τα λειτουργικά συστήματα, δεν απαιτεί μετατροπείς και χαρτογράφηση, αρκεί μόνο μία γραμμή κώδικα για την αποθήκευση αντικειμένων οποιασδήποτε πολυπλοκότητας, κ.ά.

Ωστόσο, η αντικειμενοστρεφής βάση δεδομένων db4o και τα χαρακτηριστικά της παρουσιάζουν ιδιαίτερο ενδιαφέρον στην παρούσα εργασία, με αποτέλεσμα να ακολουθεί ολόκληρο κεφάλαιο που αφιερώνεται σε αυτά. Σε αυτό το σημείο έγινε απλώς μία αναφορά στα βασικά χαρακτηριστικά που τη διέπουν.

Κεφάλαιο 1 - Η Εφαρμογή Διαχείρισης Προγράμματος Μαθημάτων

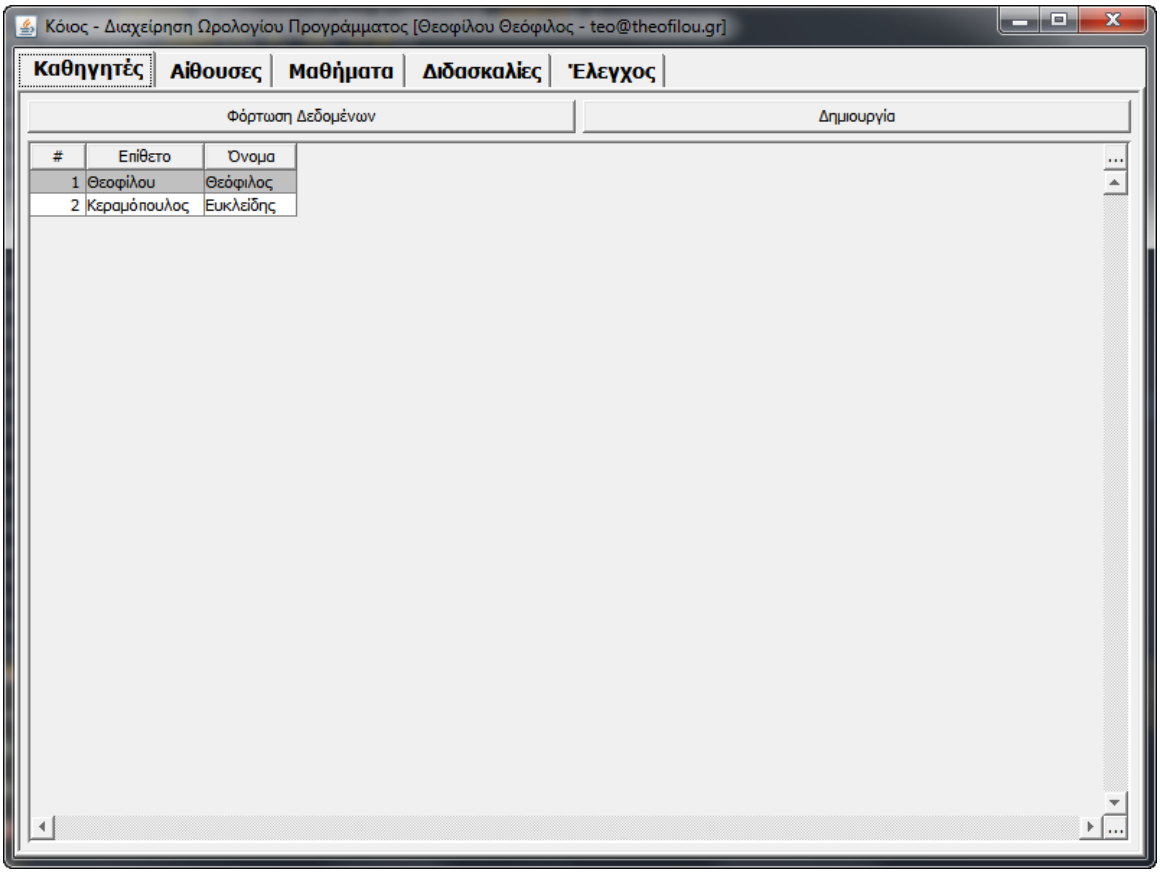
Εισαγωγή

Έπειτα από την έρευνα που πραγματοποιήθηκε για την αντικειμενοστρεφή βάση δεδομένων db4o, δημιουργήθηκε μια εφαρμογή που να τη χρησιμοποιεί. Κύριος σκοπός της εφαρμογής είναι η δημιουργία και η διαχείριση του ωρολογίου προγράμματος μαθημάτων του Τμήματος Μηχανικών Πληροφορικής ΑΤΕΙΘ, έπειτα από την εισαγωγή των κατάλληλων παραμέτρων (Καθηγητές, Αίθουσες, Μαθήματα, Τμήματα, κλπ.) και τον ορισμό των κατάλληλων περιορισμών (π.χ. ώρες διδασκαλίας μαθήματος).

Το παρόν κεφάλαιο αποτελεί έναν οδηγό χρήσης του προγράμματος που αναπτύχθηκε. Επίσης, θα παρουσιάσουμε το Μοντέλο Οντοτήτων που χρησιμοποιήθηκε. Το πώς η db4o επηρέασε και συνέβαλε την ανάπτυξη της εν λόγω εφαρμογής θα αναφερθεί σε επόμενο κεφάλαιο.

1.1 Έναρξη Εφαρμογής

Η εφαρμογή εκκινεί όπως όλες οι εφαρμογές Java. Με εκτέλεση του αρχείου «`Koios.jar`» ή εναλλακτικά με εκτέλεση της εντολής, από μια γραμμή εντολών ή το τερματικό, «`java -jar Koios.jar`» μέσα από τον φάκελο που βρίσκεται το αρχείο. Μαζί στον ίδιο φάκελο πρέπει να βρίσκονται δύο υποφάκελοι. Ο υποφάκελος «`_db`», που περιέχει δυο αρχεία («`DB.db4o`» και «`db-texts.db4o`»), και ο υποφάκελος «`lib`» που περιέχει το αρχείο «`db4o-8.0.184.15484-all-java5.jar`». Η εφαρμογή (μέσω της Java) χρειάζεται να έχει δικαιώματα και δυνατότητα εγγραφής στα αρχεία του φακέλου «`_db`». Αφού εξασφαλιστούν όλα τα παραπάνω, η εφαρμογή θα εκτελεστεί και θα δούμε το παρακάτω παράθυρο:

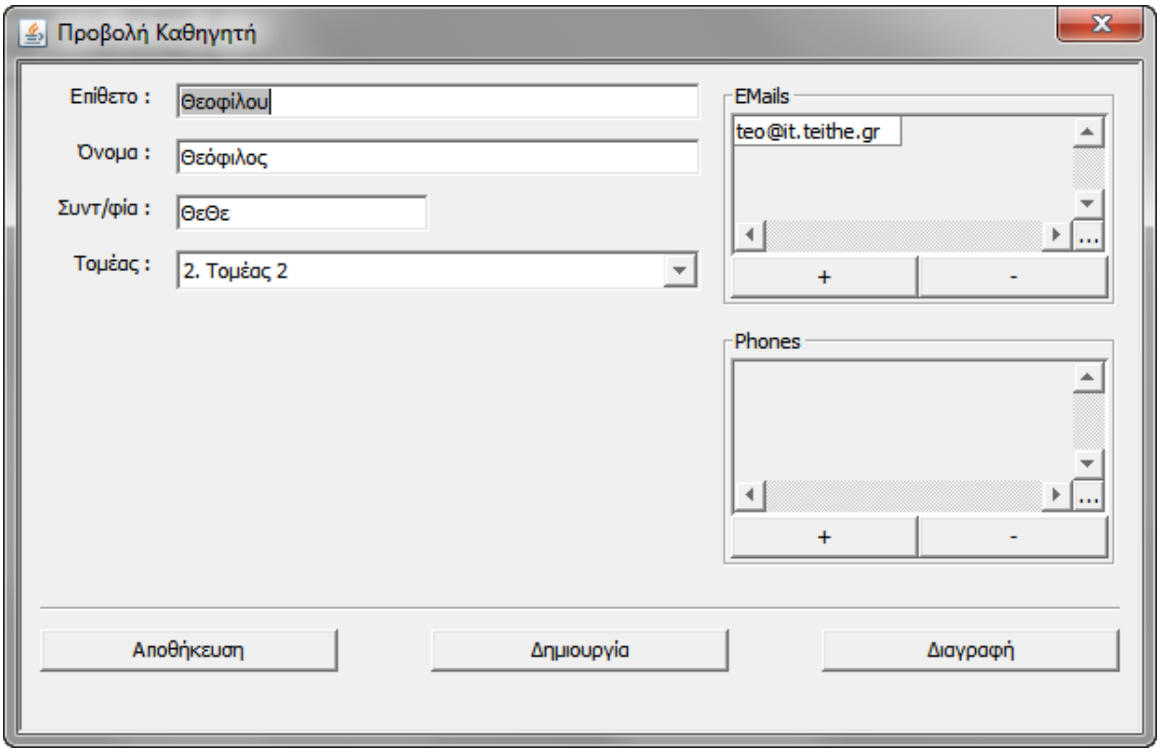


Σχήμα 1 Η Εφαρμογή Διαχείρισης Προγράμματος Μαθημάτων

Οι λειτουργίες της εφαρμογής ομαδοποιούνται σε πέντε καρτέλες. Αναλυτικότερα:

1.2 Καρτέλα Καθηγητές

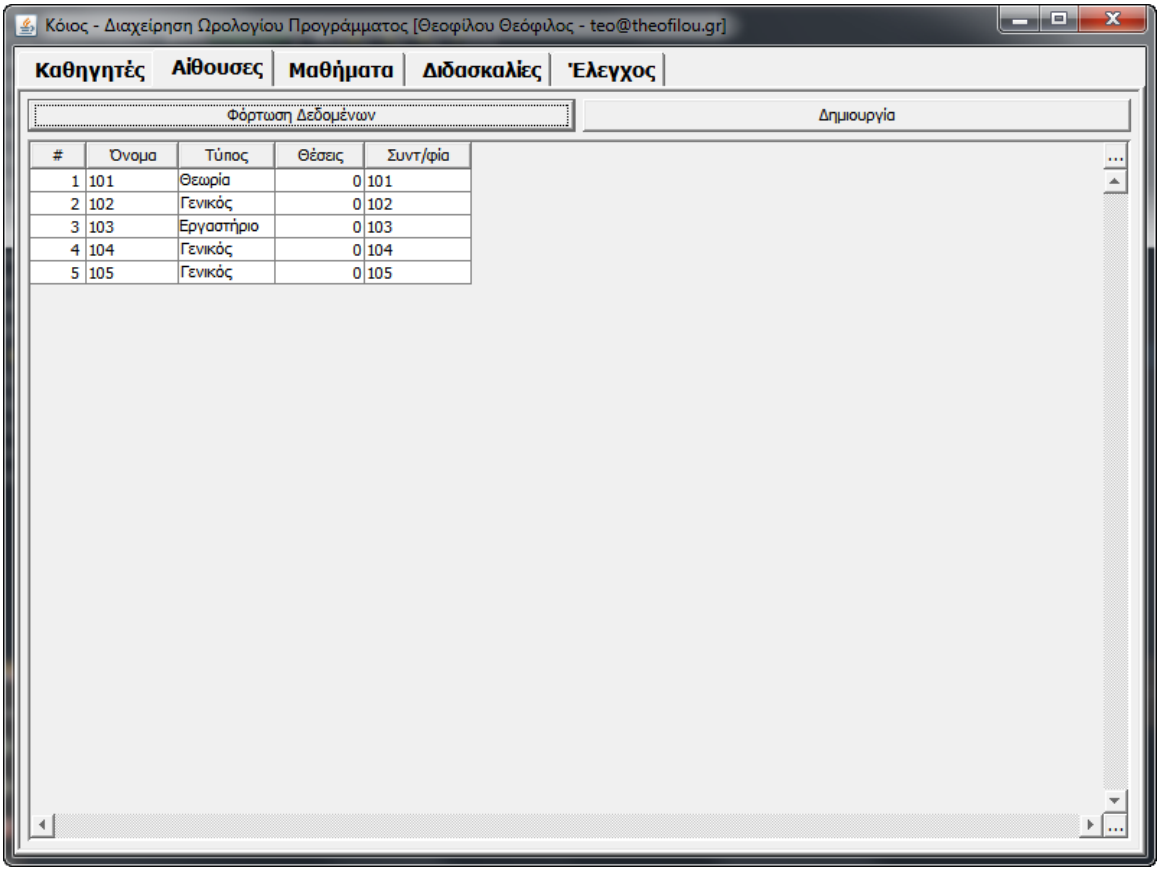
Στην καρτέλα «Καθηγητές» επεξεργαζόμαστε τους καθηγητές που υπάρχουν διαθέσιμοι για τις ανάγκες της εφαρμογής. Αποτελείται από τρία στοιχεία: έναν πίνακα και δύο κουμπιά. Ο πίνακας παρουσιάζει όλους τους Καθηγητές που έχουμε αποθηκεύσει στη βάση δεδομένων. Πατώντας το κουμπί «Φόρτωση Δεδομένων» γίνεται ανάκτηση των δεδομένων και αντίστοιχη εμφάνιση των στοιχείων στον πίνακα. Με το κουμπί «Δημιουργία» ανοίγει ένα νέο παράθυρο με το οποίο γίνεται πρόσθεση καινούριου Καθηγητή στη βάση. Πατώντας δύο φορές (διπλό κλικ) σε μια γραμμή του πίνακα Καθηγητών εμφανίζεται ένα νέο παράθυρο που περιέχει πρόσθετες λεπτομέρειες σχετικά με τον αντίστοιχο Καθηγητή και επιτρέπει στον χρήστη την επεξεργασία τους. Και στις δύο περιπτώσεις που αναφέρθηκαν το παράθυρο που εμφανίζεται είναι το παρακάτω:



Σχήμα 2 Προβολή Καθηγητή

Στα πεδία κειμένου με ετικέτα «Επίθετο», «Όνομα», «Συντ/φία» (Συντομογραφία) ο χρήστης πληκτρολογεί τα αντίστοιχα δεδομένα. Οι επιλογές στο πεδίο «Τομέας» καθορίζονται από την παραμετροποίηση του προγράμματος (από τη βάση στο αρχείο `db-texts.db4o`). Τα πλαίσια «Emails» και «Phones» χρησιμοποιούνται για την προσθήκη στοιχείων e-mail και τηλεφώνων αντίστοιχα. Με τα «+» και «–» γίνεται πρόσθεση ή αφαίρεση γραμμών στον πίνακα και με διπλό κλικ επεξεργασία αυτών.

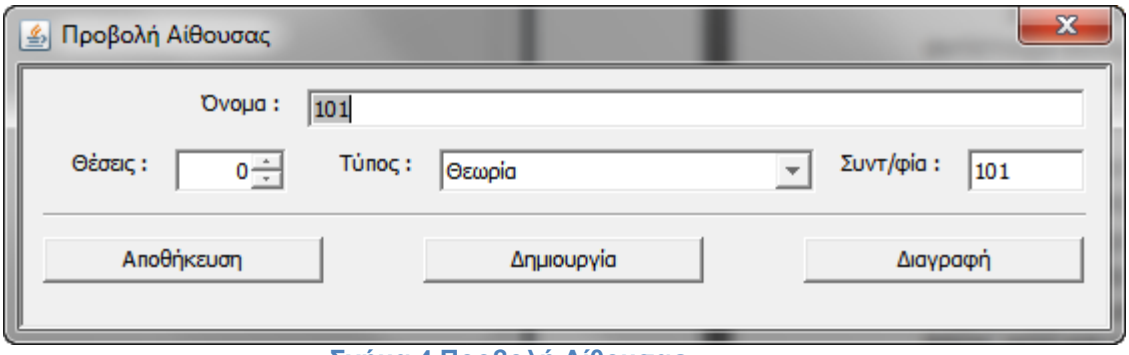
Αφού εισαχθούν ή τροποποιηθούν τα διάφορα στοιχεία, πατώντας το κουμπί «Αποθήκευση» η εφαρμογή δημιουργεί τα αντίστοιχα αντικείμενα ή αντίστοιχα τα τροποποιεί. Έπειτα τα αποθηκεύει στην βάση. Επίσης, ανανεώνει το παράθυρο με τα δεδομένα που είναι αποθηκευμένα στη βάση, για να σιγουρευτεί ο χρήστης ότι αποθηκεύτηκαν καλώς. Κανένα αντικείμενο δε δημιουργείται στη βάση, ούτε και τροποποιείται αν δεν πατηθεί το κουμπί «Αποθήκευση». Πατώντας το κουμπί «Δημιουργία» η εφαρμογή κλείνει το τρέχον παράθυρο και ανοίγει ένα καινούργιο με το οποίο μπορούμε να δημιουργήσουμε έναν καινούργιο καθηγητή. Το κουμπί «Διαγραφή» διαγράφει τον συγκεκριμένο καθηγητή από τη βάση. Το κλείσιμο του παραθύρου με το κουμπί «Χ» δεν προκαλεί καμία τροποποίηση στην βάση.



Σχήμα 3 Καρτέλα Αίθουσες

1.3 Αίθουσες

Στην καρτέλα αυτή γίνεται επεξεργασία των διαθέσιμων αιθουσών διδασκαλίας για τις ανάγκες της εφαρμογής. Η στήλη «Τύπος» εμφανίζει τιμές που καθορίζονται από την παραμετροποίηση της εφαρμογής (από την βάση στο αρχείο `db-texts.db4o`). Υπάρχουν τρία στοιχεία: ένας πίνακας και δύο κουμπιά. Ο πίνακας παρουσιάζει όλες τις αίθουσες που έχουν αποθηκευτεί στη βάση δεδομένων. Πατώντας το κουμπί «Φόρτωση Δεδομένων» γίνεται ανάκτηση των δεδομένων και εμφάνιση των στοιχείων στον πίνακα. Με το κουμπί «Δημιουργία» ανοίγει ένα νέο παράθυρο με το οποίο γίνεται πρόσθεση μιας καινούριας - αίθουσας. Πατώντας δύο φορές (διπλό κλικ) σε μια γραμμή του πίνακα εμφανίζεται ένα νέο παράθυρο που περιέχει και άλλες πρόσθετες λεπτομέρειες σχετικά με την αντίστοιχη αίθουσα και επιτρέπει την επεξεργασία αυτών. Και στις δύο περιπτώσεις το παράθυρο που εμφανίζεται είναι το παρακάτω:

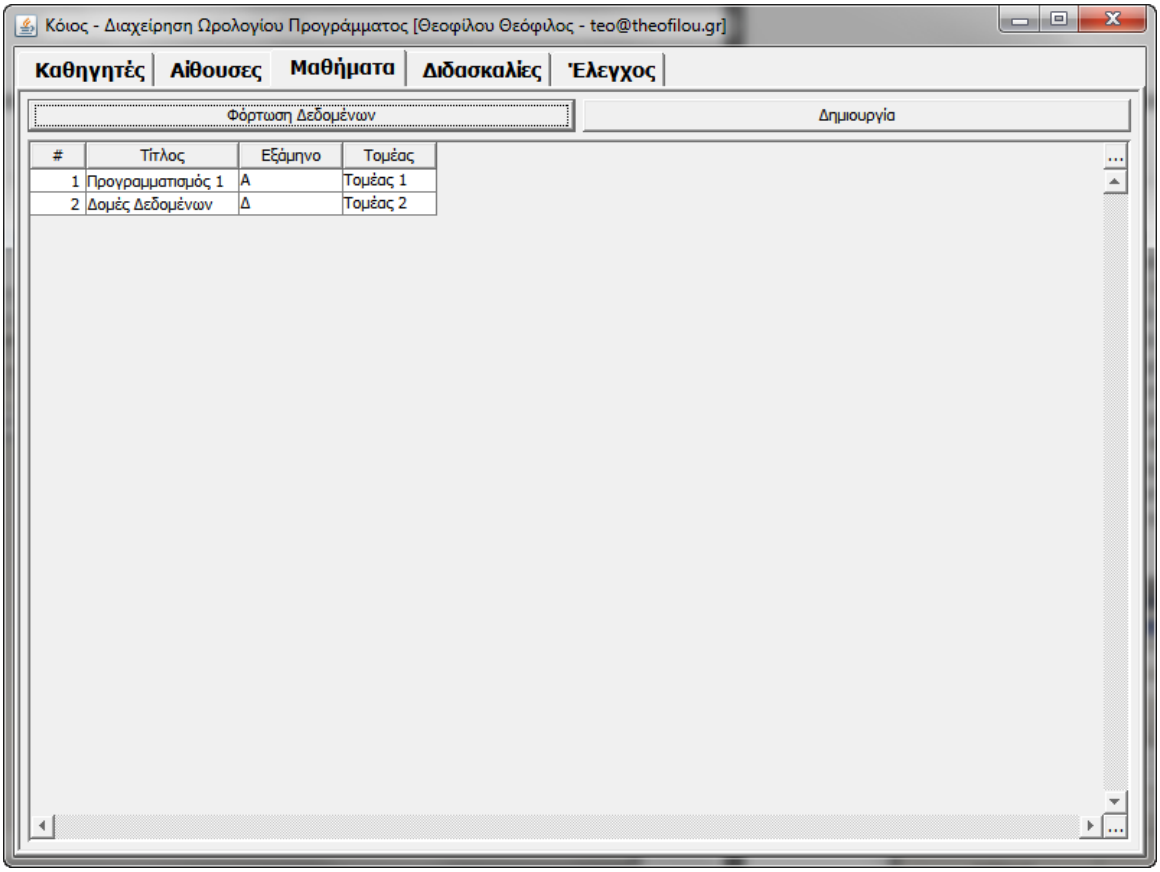


Σχήμα 4 Προβολή Αίθουσας

Στο πεδίο κειμένου «Όνομα» και «Συντ/φία» (Συντομογραφία) εισάγονται τα αντίστοιχα δεδομένα. Το πεδίο «Θέσεις» λαμβάνει ακέραιες τιμές από 0 μέχρι 200 και οι τιμές του πεδίου «Τύπος» καθορίζονται από την παραμετροποίηση (από την βάση στο αρχείο `db-texts.db4o`).

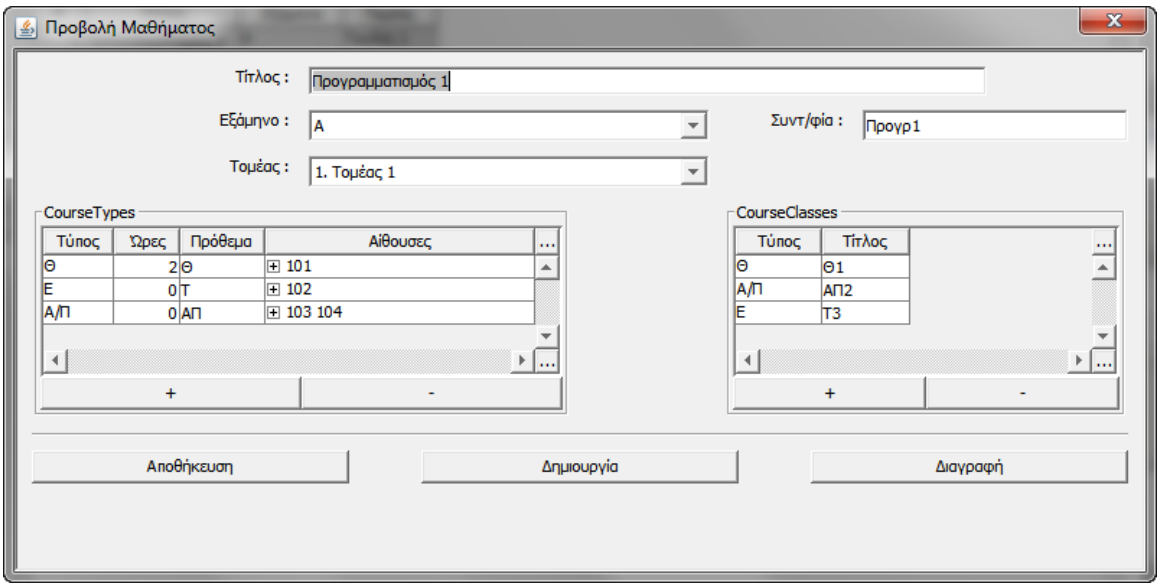
Αφού εισαχθούν ή τροποποιηθούν τα διάφορα στοιχεία, πατώντας το κουμπί «Αποθήκευση» η εφαρμογή δημιουργεί τα αντίστοιχα αντικείμενα ή αντίστοιχα τα τροποποιεί. Έπειτα τα αποθηκεύει στη βάση. Επίσης, ανανεώνει το παράθυρο με τα δεδομένα που είναι αποθηκευμένα στη βάση, για να σιγουρευτεί ο χρήστης ότι αποθηκεύτηκαν καλώς. Κανένα αντικείμενο δε δημιουργείται στη βάση, ούτε και τροποποιείται αν δεν πατηθεί το κουμπί «Αποθήκευση». Πατώντας το κουμπί «Δημιουργία» η εφαρμογή κλείνει το τρέχον παράθυρο και ανοίγει ένα καινούριο με το οποίο δημιουργείται μια καινούρια αίθουσα. Το κουμπί «Διαγραφή» διαγράφει τη συγκεκριμένη αίθουσα από τη βάση. Το κλείσιμο του παραθύρου με το κουμπί «Χ» δεν προκαλεί καμία τροποποίηση στη βάση.

1.5 Μαθήματα



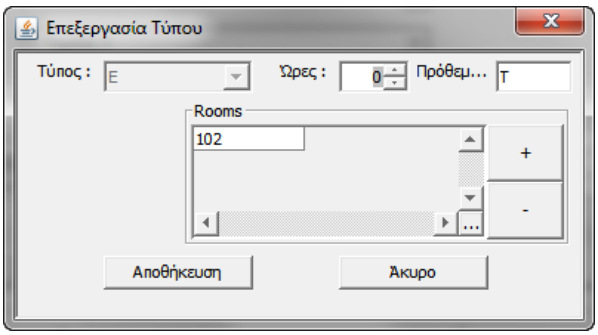
Σχήμα 5 Καρτέλα Μαθήματα

Σ' αυτήν την καρτέλα γίνεται επεξεργασία των μαθημάτων που είναι διαθέσιμα. Οι στήλες «Εξάμηνο» και «Τομέας» εμφανίζουν τιμές που καθορίζονται από την παραμετροποίηση της εφαρμογής (από την βάση στο αρχείο db-texts.db4o). Αποτελείται από τρία στοιχεία: έναν πίνακα και δύο κουμπιά. Ο πίνακας παρουσιάζει όλα τα μαθήματα που έχουν αποθηκευτεί στη βάση δεδομένων. Πατώντας το κουμπί «Φόρτωση Δεδομένων» γίνεται ανάκτηση των δεδομένων και εμφάνιση των στοιχείων στον πίνακα. Με το κουμπί «Δημιουργία» ανοίγει ένα νέο παράθυρο με το οποίο γίνεται πρόσθεση ενός καινούριου μαθήματος. Πατώντας δύο φορές (διπλό κλικ) σε μια γραμμή του πίνακα εμφανίζεται ένα νέο παράθυρο που επιτρέπει την προβολή επιπρόσθετων πληροφοριών σχετικά με το αντίστοιχο μάθημα όπως και την επεξεργασία αυτών. Και στις δύο περιπτώσεις το παράθυρο που εμφανίζεται είναι το παρακάτω:



Σχήμα 6 Προβολή Μαθήματος

Για τα πεδία «Εξάμηνο» και «Τομέας», οι τιμές καθορίζονται από την παραμετροποίηση της εφαρμογής (η βάση στο αρχείο `db-texts.db4o`). Το πλαίσιο που καθορίζεται με το τίτλο «Τύποι Διδασκαλίας» χρησιμοποιείται για την κατηγοριοποίηση των τμημάτων διδασκαλίας. Οι κατηγορίες (τύποι) καθορίζονται από την παραμετροποίηση της εφαρμογής (η βάση στο αρχείο db-texts.db4o). Κάθε γραμμή του πίνακα αναφέρεται μόνο σε ένα τύπο και δε μπορεί να υπάρξουν δύο γραμμές που αναφέρονται στον ίδιο τύπο. Κάθε τύπος έχει ως παραμέτρους το πλήθος των ωρών διδασκαλίας ανά βδομάδα (στήλη «Ώρες»), το πρόθεμα που θα έχει ο τίτλος του κάθε τμήματος αυτού (στήλη «Πρόθεμα») και τέλος σε ποιες αίθουσες μπορεί να πραγματοποιηθεί αυτός ο τύπος διδασκαλίας (στήλη «Αίθουσες»). Πατώντας το κουμπί με το «+» γίνεται πρόσθεση μιας καινούριας γραμμής, αρκεί να υπάρχουν διαθέσιμες επιλογές, με βάση την παραμετροποίηση της εφαρμογής. Με το κουμπί «–» αφαιρείται η επιλεγμένη γραμμή. Επειδή, όπως αναφέρεται και παρακάτω, τα τμήματα που ορίζονται εξαρτώνται από το αν έχει ορισθεί ο αντίστοιχος τύπος διδασκαλίας, η αφαίρεση ενός τύπου, καταργεί και όλα τα σχετικά τμήματα.



Σχήμα 7 Επεξεργασία Τύπου

Πατώντας είτε το κουμπί «+», είτε κάνοντας διπλό κλικ σε μια γραμμή, το παράθυρο που εμφανίζεται είναι το παραπάνω. Οι μόνες διαφορές ανάμεσα στα δύο είναι ότι στην επεξεργασία, το πλαίσιο επιλογής του τύπου δεν επιτρέπει στον

χρήστη να αλλάξει την επιλογή και το κουμπί «Προσθήκη» γίνεται «Αποθήκευση». Το πλαίσιο επιλογής «Τύπος» περιέχει τιμές που καθορίζονται από την παραμετροποίηση της εφαρμογής. Στο πλαίσιο «Ώρες» εισάγεται μια θετική ακέραια τιμή που αντιστοιχεί στις ώρες διδασκαλίας του συγκεκριμένου τύπου ανά εβδομάδα. Στο πλαίσιο «Πρόθεμα» εισάγεται το πρόθεμα που θα έχει ο τίτλος των αντίστοιχων τμημάτων. Στο πλαίσιο με τίτλο «Αίθουσες» ορίζεται σε ποιες αίθουσες μπορεί να διδάσκεται ο συγκεκριμένος τύπος διδασκαλίας. Αν δεν έχει ορισθεί τίποτα, τότε η εφαρμογή δεν θα εμφανίζει τα αντίστοιχα τμήματα πουθενά για να μπορεί να γίνει ανάθεσή τους σε συγκεκριμένη ώρα, αίθουσα ή καθηγητές. Πατώντας το κουμπί «+» γίνεται πρόσθεση αιθουσών στη λίστα, μέσω ενός παραθύρου που ζητά από τον χρήστη να επιλέξει μια από τις αίθουσες που είναι αποθηκευμένες στη βάση και δεν είναι στη λίστα. Το κουμπί «–» αφαιρεί την επιλεγμένη αίθουσα από τη λίστα. Αν δεν είναι τίποτα επιλεγμένο, τότε τίποτα δεν αφαιρείται.

Το πλαίσιο «Τμήματα» χρησιμεύει για την προβολή και επεξεργασία των τμημάτων που υπάρχουν για το τρέχον μάθημα. Πατώντας το κουμπί «+» γίνεται πρόσθεση ενός καινούριου τμήματος, με το «–» αφαιρείται το επιλεγμένο (αν δεν υπάρχει τίποτα επιλεγμένο, δεν αφαιρείται τίποτα) και με διπλό κλικ γίνεται επεξεργασία του επιλεγμένου τμήματος. Το παράθυρο δημιουργίας / επεξεργασίας είναι το παρακάτω:

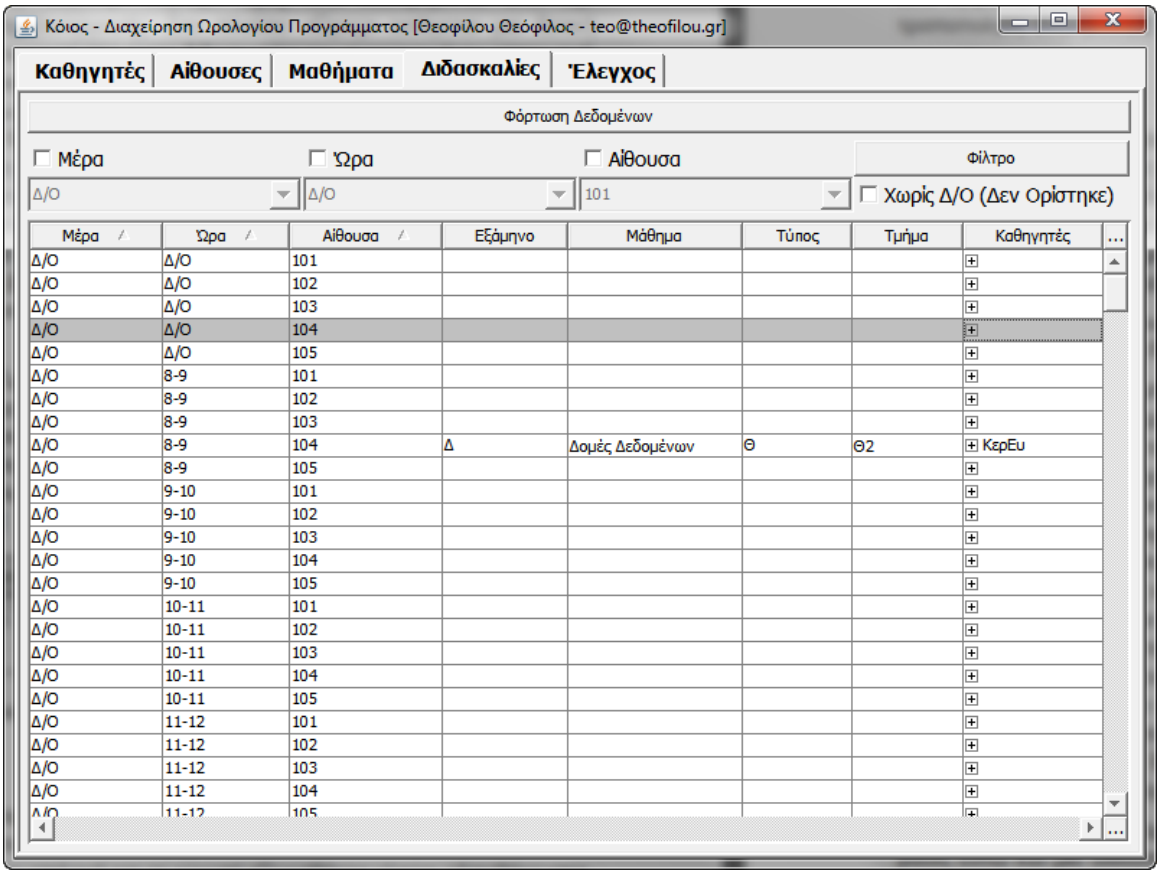
Σχήμα 8 Επεξεργασία Τμήματος

Οι μόνες διαφορές ανάμεσα στην προσθήκη και στην επεξεργασία είναι ότι στην επεξεργασία, το πλαίσιο επιλογής του τύπου δε δίνει τη δυνατότητα στον χρήστη να αλλάξει την επιλογή και το κουμπί «Προσθήκη» γίνεται «Αποθήκευση». Στο μεσαίο «γκριζαρισμένο» πλαίσιο φαίνεται το πρόθεμα που έχει ορισθεί για τον τίτλο του συγκεκριμένου τύπου, ενώ το δεξιό πλαίσιο κειμένου χρησιμεύει για τη συμπλήρωση του τίτλου (π.χ. ένας αριθμός).

Αφού γίνει η εισαγωγή ή η τροποποίηση των διάφορων στοιχείων, πατώντας το κουμπί «Αποθήκευση» η εφαρμογή δημιουργεί τα αντίστοιχα αντικείμενα ή αντίστοιχα τα τροποποιεί. Έπειτα τα αποθηκεύει στην βάση. Επίσης, ανανεώνει το παράθυρο με τα δεδομένα που είναι αποθηκευμένα στη βάση, για να σιγουρευτεί ο χρήστης ότι αποθηκεύτηκαν καλώς. Κανένα αντικείμενο δεν δημιουργείται στην βάση, ούτε και τροποποιείται αν δεν πατήσουμε το κουμπί «Αποθήκευση». Πατώντας το κουμπί «Δημιουργία» η εφαρμογή κλείνει το τρέχον παράθυρο και ανοίγει ένα καινούριο με το οποίο μπορεί να δημιουργηθεί ένα

καινούριο μάθημα. Το κουμπί «Διαγραφή» διαγράφει το συγκεκριμένο μάθημα από τη βάση. Το κλείσιμο του παραθύρου με το κουμπί «Χ» δεν προκαλεί καμία τροποποίηση στην βάση.

1.6 Διδασκαλίες



Σχήμα 9 Καρτέλα Διδασκαλίες

Αφού έχουν εισαχθεί και επεξεργαστεί όλα τα συστατικά που χρειάζονται για να τη δημιουργία του προγράμματος μαθημάτων, μπορεί να γίνει μετάβαση στην καρτέλα «Διδασκαλίες». Η εν λόγω καρτέλα αποτελείται από τρία μέρη: ένα κουμπί «Φόρτωση Δεδομένων», έναν πίνακα για την προβολή και επεξεργασία του προγράμματος μαθημάτων και ανάμεσα είναι ένα σύνολο στοιχείων με τα οποία μπορούν να φιλτραριστούν τα δεδομένα που φαίνονται στον πίνακα.

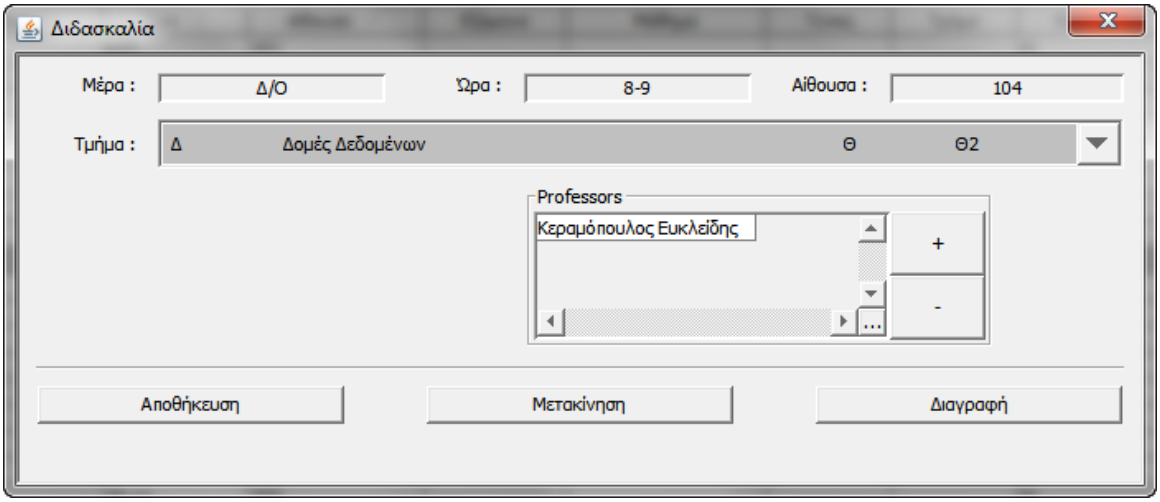
Το κουμπί «Φόρτωση Δεδομένων» διαβάζει τις διδασκαλίες που είναι αποθηκευμένες στη βάση δεδομένων και τις παρουσιάζει στον πίνακα.

Τα φίλτρα που μπορούν να εφαρμοστούν στην προβολή είναι ένας συνδυασμός ενός ή περισσοτέρων από τις επιλογές «Μέρα», «Ώρα» και «Αίθουσα». Επιλέγοντας το αντίστοιχο κουτάκι καθορίζεται αν θα συμμετέχει στο φιλτράρισμα. Όταν μαρκαριστεί ένα κουτάκι δίνεται η δυνατότητα να επιλεχθεί από τη λίστα κάποια από τις αντίστοιχες τιμές. Οι μέρες και οι ώρες καθορίζονται από την παραμετροποίηση της εφαρμογής (η βάση στο αρχείο `db-texts.db4o`).Οι

αίθουσες είναι οι αίθουσες που υπάρχουν αποθηκευμένες στη βάση μας. Το «Δ/Ο» στις μέρες και στις ώρες σημαίνει Δεν Ορίστηκε. Υπάρχει η δυνατότητα να οριστούν διδασκαλίες που να μη γνωρίζει ακόμα ο χρήστης σε ποια μέρα ή/και ώρα να τις ορίσει. Αυτό το σκοπό εξυπηρετεί αυτό το «Δ/Ο». Αν μαρκαριστεί το κουτάκι «Χωρίς Δ/Ο (Δεν Ορίστηκε)» το φίλτρο θα αποκρύψει γραμμές που περιέχουν είτε στη Μέρα είτε στην Ώρα το «Δ/Ο». Η ενεργοποίηση του φίλτρου στον πίνακα γίνεται με το κουμπί «Φίλτρο». Δεν αποτελεί το κλασσικό κουμπί, αλλά κουμπί εναλλαγής. Εναλλάσσεται μεταξύ της κατάστασης επιλογής και μη επιλογής. Όταν είναι επιλεγμένο το φίλτρο εφαρμόζεται και αντίθετα.

Ο πίνακας για να εμφανίζει γραμμές έχει μια προϋπόθεση. Να υπάρχει στη βάση έστω και μία αίθουσα αποθηκευμένη. Οι γραμμές του πίνακα αντιστοιχούν στους δυνατούς συνδυασμούς μεταξύ Μέρας, Ώρας και διαθέσιμων Αιθουσών. Αυτό γίνεται για να εξασφαλιστεί ότι την ίδια ώρα θα γίνεται το πολύ ένα μάθημα σε μια συγκεκριμένη αίθουσα. Η στήλη «Καθηγητές» εμφανίζει τους καθηγητές που έχουν ανατεθεί σ' αυτή τη διδασκαλία. Στην προεπιλεγμένη κατάσταση τους εμφανίζει με τη συντομογραφία τους σε μια γραμμή. Κάνοντας κλικ στο τετραγωνισμένο + εμφανίζονται με πλήρη όνομα, σε πολλαπλές γραμμές. Οι υπόλοιπες στήλες εμφανίζουν τα υπόλοιπα στοιχεία του τμήματος.

Με διπλό κλικ γίνεται επεξεργασία της επιλεγμένης διδασκαλίας που υπάρχει σ' αυτήν την Μέρα/Ώρα/Αίθουσα ή δημιουργείται μια καινούρια. Το παράθυρο που εμφανίζεται και στις δυο περιπτώσεις είναι το παρακάτω:



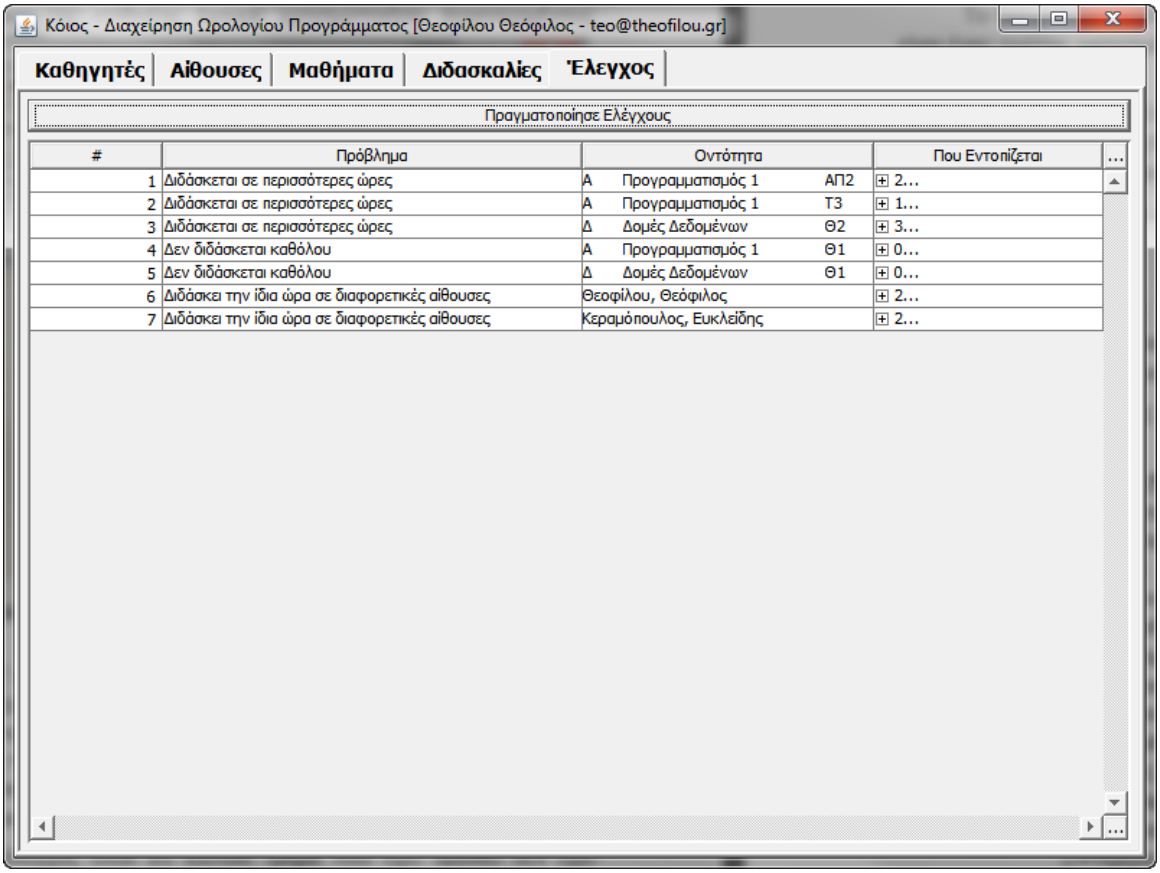
Σχήμα 10 Επεξεργασία Διδασκαλίας

Στην πρώτη γραμμή μας δείχνει την επιλεγμένη Μέρα/Ώρα/Αίθουσα που λογικά δεν τροποποιούνται. Στο πλαίσιο «Τμήμα» δίνει στον χρήστη τη δυνατότητα να επιλέξει κάποιο από τα τμήματα που μπορούν να διδαχτούν στην επιλεγμένη αίθουσα, όπως το όρισε όταν όριζε τους τύπους διδασκαλίας ενός μαθήματος. Αν για κάποια αίθουσα δεν έχει οριστεί κανένας τύπος κάποιου μαθήματος, τότε το παραπάνω παράθυρο δεν θα εμφανιστεί.

Το πλαίσιο «Καθηγητές» χρησιμεύει για να ορίζονται οι καθηγητές που διδάσκουν στη συγκεκριμένη διδασκαλία. Αν αλλάξει το τμήμα οι καθηγητές θα παραμείνουν. Με το κουμπί «+» εμφανίζεται μια λίστα με τους αποθηκευμένους και μη υπαρκτούς στην προαναφερόμενη λίστα καθηγητές. Το κουμπί «–» αφαιρεί από τη λίστα τον επιλεγμένο καθηγητή.

Αφού γίνουν όσες επεξεργασίες απαιτούνται, με το κουμπί «Αποθήκευση» αποθηκεύονται οι αντίστοιχες επιλογές. Δηλαδή συσχετίζονται το τμήμα και οι καθηγητές με τη συγκεκριμένη Μέρα/Ώρα/Αίθουσα. Με το κουμπί «Μετακίνηση» δίνεται η δυνατότητα στον χρήστη να μετακινήσει το τμήμα μαζί με τους καθηγητές που είναι συσχετισμένα στη βάση για την συγκεκριμένη Μέρα/Ώρα/Αίθουσα σε μια άλλη Μέρα/Ώρα/Αίθουσα. Η μετακίνηση μπορεί να γίνει μόνο σε συνδυασμούς που η αίθουσα είναι ορισμένη ως δυνατή αίθουσα διδασκαλίας του τύπου. Οπότε και η λίστα με επιλογές που φαίνονται, όταν πατηθεί το κουμπί, αποτελείται μόνο από αυτές. Αλλιώς δεν μπορεί να γίνει μετακίνηση. Τέλος, με το κουμπί «Διαγραφή» διαγράφεται η συγκεκριμένη διδασκαλία από τη βάση που σημαίνει ότι κανένα τμήμα και καθηγητής δεν είναι συσχετισμένα με την συγκεκριμένη Μέρα/Ώρα/Αίθουσα.

1.7 Έλεγχος



Σχήμα 11 Καρτέλα Έλεγχος

Η συγκεκριμένη καρτέλα δίνει τη δυνατότητα να ελέγξει για προβλήματα στο πρόγραμμα που έχει δημιουργηθεί μέχρι στιγμής. Πατώντας το κουμπί «Πραγματοποίησε Ελέγχους» θα εκτελέσει κάθε έλεγχο που έχει προγραμματιστεί να κάνει. Ο πίνακας παρουσιάζει που έχουν εντοπιστεί προβλήματα. Η στήλη «Πρόβλημα» δηλώνει το όνομα του προβλήματος, η στήλη «Οντότητα» το ποιος είναι αυτός που έχει το πρόβλημα, ενώ η στήλη «Πού εντοπίζεται» δείχνει σε ποιες διδασκαλίες (γραμμές στον πίνακα Διδασκαλίες) δημιουργείται το πρόβλημα. Στην προεπιλεγμένη μορφή, η στήλη περιέχει ένα νούμερο που δείχνει το πλήθος των διδασκαλιών που εντοπίζεται το πρόβλημα. Πατώντας το τετραγωνισμένο + βλέπουμε ποιες είναι οι συγκεκριμένες διδασκαλίες.

Για τις ανάγκες της επίδειξης έχουμε χρησιμοποιήσει συγκεκριμένους ελέγχους. Ο ένας εξ' αυτών είναι για το εάν ένα τμήμα διδάσκεται περισσότερες ώρες από όσες ορίζει ο αντίστοιχος τύπος διδασκαλίας του. Οπότε και η στήλη «Οντότητα» δείχνει το συγκεκριμένο τμήμα. Η στήλη «Πού εντοπίζεται» δείχνει όλες τις διδασκαλίες που εμφανίζεται το τμήμα σαν προβληματικές. Η διόρθωση γίνεται με το να επιστρέψει ο χρήστης στην καρτέλα «Διδασκαλίες» και να αναθεωρήσει τις επιλογές για το συγκεκριμένο τμήμα.

Ένας άλλος έλεγχος είναι αν κάποιο τμήμα που έχει ορισθεί δεν έχει ανατεθεί σε κάποια διδασκαλία. Η στήλη «Οντότητα» αναφέρεται σ' αυτό το τμήμα. Η στήλη «Πού εντοπίζεται» δε δίνει κάποια επιπλέον πληροφορία. Ο χρήστης χρειάζεται να μεταβεί στην καρτέλα «Διδασκαλίες» και να κάνει τις ανάλογες αναθεωρήσεις.

Τελευταίος έλεγχος είναι το αν κάποιος καθηγητής έχει ανατεθεί την ίδια μέρα και ώρα σε διαφορετικά τμήματα. Η στήλη «Οντότητα» δείχνει τον συγκεκριμένο Καθηγητή και η στήλη «Πού εντοπίζεται» το σε ποιες διδασκαλίες εμφανίζεται την ίδια μέρα και ώρα.

1.8 Διάγραμμα Κλάσεων και των Συσχετίσεών τους

Στην περίπτωση του προβλήματος που εξετάζεται στην παρούσα εργασία έχουμε τις οντότητες:

- Μάθημα
- Καθηγητής
- Αίθουσα Διδασκαλίας

Και τα χαρακτηριστικά τους είναι:

- Μάθημα
  - Τίτλος
  - Εξάμηνο

    - Συντομογραφία
- Καθηγητής
    - Επώνυμο
    - Όνομα
    - Συντομογραφία
- Αίθουσα Διδασκαλίας
    - Όνομα Αίθουσας
    - Αριθμός Θέσεων
    - Τύπος Αίθουσας
    - Συντομογραφία

Οι συσχετίσεις μεταξύ τους δημιουργούνται μέσω μιας άλλης κατηγορίας οντοτήτων, τις Ιδιότητες. Κάθε Ιδιότητα αναφέρεται σε μια από τις προηγούμενες οντότητες ή σε κάποια άλλη Ιδιότητα. Η Ιδιότητα περιέχει ένα αντικείμενο το οποίο είναι η τιμή της Ιδιότητας. Οπότε τις Ιδιότητες τις χρησιμοποιούμε είτε για να προσθέσουμε κάποια χαρακτηριστικά σε μια οντότητα είτε για να δημιουργήσουμε συσχετίσεις.

Αυτές οι ιδιότητες, με τα χαρακτηριστικά τους, είναι :

- Τηλέφωνο
    - Ο αριθμός τηλεφώνου που έχει ο Καθηγητής
- Email
    - Το e-mail που έχει ο Καθηγητής
- Τομέας
    - Ο Τομέας που ανήκει ο Καθηγητής ή το Μάθημα
- Τύπος Διδασκαλίας
    - Ο τύπος της διδασκαλίας ενός Μαθήματος
- Αίθουσα που μπορεί να διδαχθεί ο Τύπος Διδασκαλίας
    - Αίθουσα
- Τμήμα
    - Το όνομα ενός τμήματος ενός Τύπου Διδασκαλίας
- Διδασκαλία για το που και πότε διδάσκεται ένα Τμήμα
    - Δωμάτιο
    - Μέρα
    - Ώρα
- Καθηγητές που διδάσκουν μια διδασκαλία
    - Καθηγητής

Είδη συσχετίσεων:

- Κάθε Μάθημα έχει κάποιους Τύπους Διδασκαλίας.
- Κάθε Μάθημα ανήκει σε κάποιους Τομείς.
- Κάθε Τύπος Διδασκαλίας μπορεί να διδάσκεται κάποιες Αίθουσες.
- Κάθε Τύπος Διδασκαλίας μπορεί να έχει κάποια Τμήματα.
- Κάθε Τμήμα μπορεί να διδάσκεται σε κάποιες Διδασκαλίες.
- Κάθε Διδασκαλία μπορεί να διδάσκεται από κάποιους Καθηγητές.
- Κάθε Καθηγητής έχει κάποια τηλέφωνα.
- Κάθε Καθηγητής έχει κάποια e-mail.

Κεφάλαιο 2 - Οδηγός της db4o

Εισαγωγή

Η db4o είναι μία αντικειμενοστρεφής βάση δεδομένων, ένας τρόπος για να μπορούν να γίνουν τα αντικείμενα μόνιμα (persistence - να παραμένουν αναλλοίωτα μεταξύ διαφορετικών εκτελέσεων του κώδικα). Απαιτούνται μονάχα μερικά λεπτά για την εγκατάστασή της και είναι έτοιμη να χρησιμοποιηθεί στην υποψήφια εφαρμογή. Αξίζει να σημειωθεί ότι χρησιμοποιείται όπως θα χρησιμοποιούταν μια δομή δεδομένων. Ο προγραμματιστής, χωρίς να χρειάζεται ν' ασχοληθεί με πολλές λεπτομέρειες, έχει με λίγες μόνο γραμμές κώδικα, πρόσβαση σε μια βάση δεδομένων. Μια βάση δεδομένων που ακολουθεί το αντικειμενοστρεφές μοντέλο ανάπτυξης και έτσι δε θα χρειαστεί να «αλλάζει λογικές», για να τη διαχειριστεί. Ο προγραμματιστής ακολουθεί επακριβώς το «Μοντέλο Οντοτήτων», χωρίς να πρέπει να πραγματοποιήσει μετατροπές ή αντιστοιχήσεις που να ικανοποιούν τη βάση δεδομένων.

Το κεφάλαιο αυτό αποτελεί ένα μικρό οδηγό της db4o. Δεν έχει το σκοπό πλήρους εγχειριδίου, αλλά να δείξει πώς μπορεί να χρησιμοποιηθεί σε μια βασική εφαρμογή. Όλες οι δυνατότητες της db4o είναι αδύνατο να αναφερθούν στην παρούσα εργασία. Ενδεικτικά, το επίσημο εγχειρίδιο της db4o αποτελείται από 321 σελίδες. Σ' αυτό το κεφάλαιο θα δούμε τα βασικά σημεία και τις λειτουργίες της συγκεκριμένης βάσης.

2.1 Μια πρώτη εικόνα

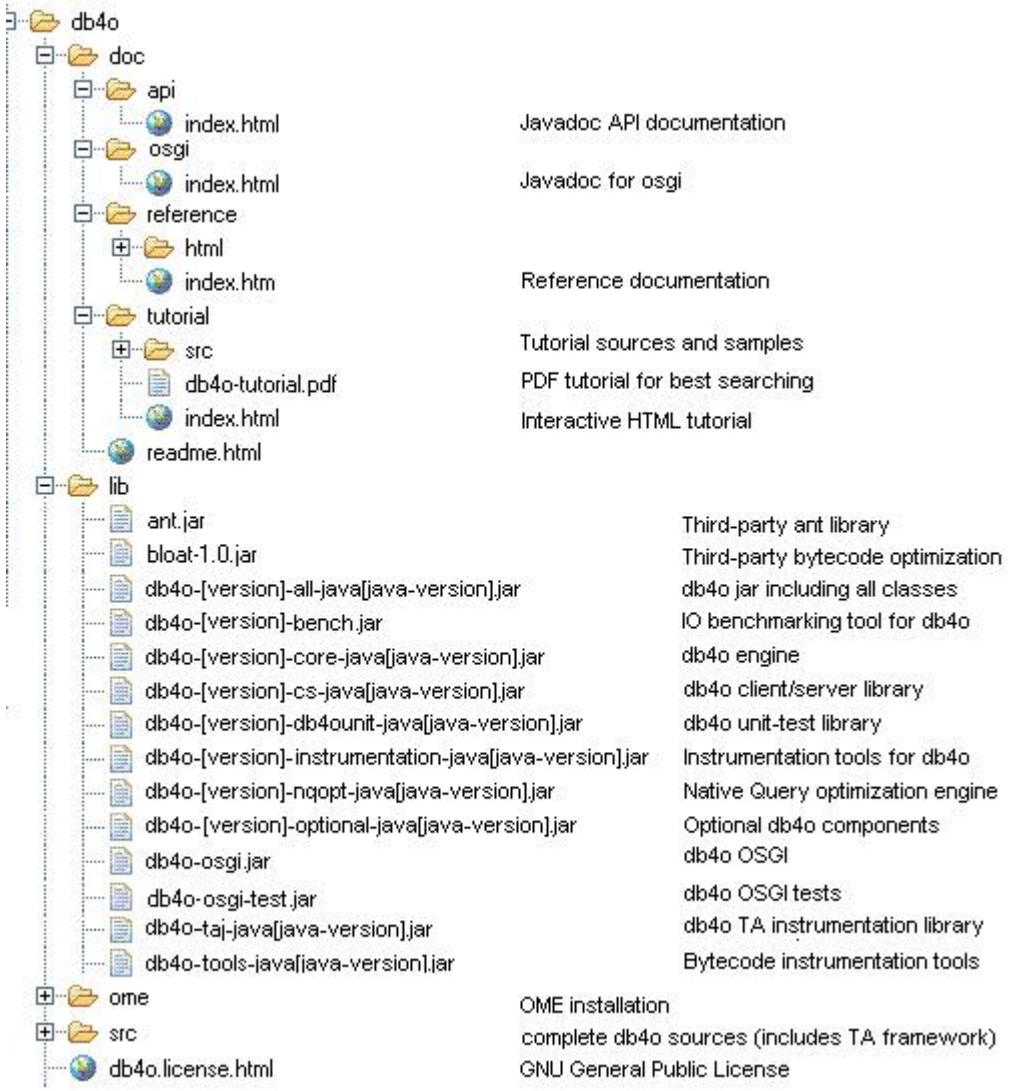
Η db4o, όπως έχει ήδη αναφερθεί, αποθηκεύει αντικείμενα. Τα αντικείμενα έχουν κατάσταση και λειτουργίες ή αλλιώς, δεδομένα και μεθόδους. Για να γίνει ένα αντικείμενο μόνιμο, χρειάζεται να αποθηκευτούν κάπου τα δεδομένα και αυτά να υπάρχουν επακριβώς, όταν θα επαναχρησιμοποιηθεί το αντικείμενο σε επόμενη εκτέλεση. Ζητώντας από την db4o να αποθηκεύσει ένα αντικείμενο τύπου Α, διαβάζει τα δεδομένα του (τις τιμές των μεταβλητών, άσχετα με το επίπεδο πρόσβασης) και τα αποθηκεύει σημειώνοντας τα ως ένα αντικείμενο τύπου Α. Και όταν ανακτηθεί ένα αντικείμενο, η db4o φροντίζει να κατανεμηθεί χώρος στη μνήμη και τοποθετεί τα δεδομένα που υπήρχαν, όταν έγινε η αποθήκευση. Με λίγα λόγια, μεταφέρει την περιοχή της μνήμης που καταλαμβάνει το αντικείμενο στη βάση δεδομένων, κατά την αποθήκευση, και αντίστροφα, κατά την ανάκτηση.

Η προσπέλαση στην βάση δεδομένων γίνεται με τη δημιουργία ενός αντικειμένου τύπου `ObjectContainer`. Το αντικείμενο αυτό είναι που δημιουργεί μια συνεδρία προς τη βάση δεδομένων. Ο `ObjectContainer`, από την έναρξή του ως το κλείσιμό του, παρακολουθεί τη μνήμη και ξέρει ποια αντικείμενα σ' αυτήν αντιστοιχούν σε αντικείμενα στη βάση δεδομένων. Μ' αυτόν τον τρόπο, η αποθήκευση ενός τροποποιημένου αντικειμένου, έχει ως αποτέλεσμα την

ανανέωσή του στη βάση δεδομένων και όχι τη δημιουργία κάποιου άλλου. Ανάλογη μέριμνα υπάρχει και για τις περιπτώσεις που πεδία ενός αντικειμένου αναφέρονται σε άλλα αντικείμενα. Με την αποθήκευση / τροποποίηση δε χάνονται ή τροποποιούνται αυτές οι αναφορές.

2.2 Εγκατάσταση

Η εγκατάσταση της db4o είναι πολύ απλή. Αρχικά, ο χρήστης μεταβαίνει στην ιστοσελίδα www.db4o.com, κατεβάζει την έκδοση για την γλώσσα προγραμματισμού που τον ενδιαφέρει (στην περίπτωσή μας, για την Java). Αποσυμπιέζει το αρχείο που κατέβασε και η δομή των αρχείων είναι ανάλογη με την παρακάτω εικόνα.



Σχήμα 12 Η δομή του φακέλου της db4o

Για την εγκατάσταση, αυτό που χρειάζεται είναι ο φάκελος `lib`. Μέσα σ' αυτόν περιέχονται διάφορες βιβλιοθήκες (`*.jar`) έτοιμες προς χρήση. Η βιβλιοθήκη που χρειάζεται είναι η `db4o-<έκδοση της db4o>-core-`

`java<έκδοση της Java>.jar`. Αυτή περιέχει όλα τα βασικά στοιχεία της db4o και λογικά θα καλύψει τις προκύπτουσες ανάγκες. Για την κάλυψη κάθε ενδεχόμενου θα μπορούσε να χρησιμοποιηθεί το `db4o-<έκδοση της db4o>-all-java<έκδοση της Java>.jar` που περιέχει όλες τις δυνατότητές της. Αφού επιλεχθεί η βιβλιοθήκη δηλώνεται στην εφαρμογή, όπως όλες οι άλλες βιβλιοθήκες, δηλαδή δηλώνεται στο CLASSPATH. Τέλος, γίνεται import των πακέτων ή των κλάσεων που χρειάζονται στα αρχεία Java που θα χρησιμοποιηθεί.

Μια πολύ καλή πρακτική είναι η δημιουργία στην εφαρμογή μιας κλάσης που θα έχει το ρόλο της άμεσης προσπέλασης στη βάση. Έτσι θα δύναται να ελέγχεται ότι υπάρχει μοναδική σύνδεση από την εφαρμογή στη βάση, αλλά και θα μπορούν να εποπτεύονται όλες οι λειτουργίες. Όπως προαναφέρθηκε, προσπελαύνουμε τη βάση δημιουργώντας ένα αντικείμενο τύπου `ObjectContainer` και καλώντας τις μεθόδους `store()`, `delete()` και `query()` αυτού του αντικειμένου. Στη συνέχεια θα δούμε στην πράξη πώς λειτουργούν όλα αυτά.

2.3 Βασικές Λειτουργίες db4o

Για τους σκοπούς της επίδειξης των λειτουργιών της db4o στο παρών κεφάλαιο, θα χρησιμοποιήσουμε την παρακάτω κλάση για την δημιουργία αντικειμένων που θα αποθηκεύουμε στην βάση δεδομένων.

```
public class Pilot {
    private String name;
    private int points;

    public Pilot(String name,int points) {
        this.name=name;
        this.points=points;
    }

    public int getPoints() {
        return points;
    }

    public void addPoints(int points) {
        this.points+=points;
    }

    public String getName() {
        return name;
    }

      @Override
    public String toString() {
        return name+"/"+points;
    }
}
```

Σχήμα 13 Η κλάση Pilot

Πρόσβαση στη Βάση

Για να αποκτηθεί πρόσβαση σε ένα αρχείο βάσης δεδομένων db4o ή να δημιουργηθεί ένα νέο, καλείται η μέθοδος `Db4oEmbedded.openFile()` που παίρνει δύο παραμέτρους. Ένα αντικείμενο τύπου `EmbeddedConfiguration`, που περιέχει ένα σύνολο ρυθμίσεων, και μια συμβολοσειρά με τη διαδρομή προς το αρχείο της βάσης δεδομένων. Το αντικείμενο με τις ρυθμίσεις, στην απλούστερη μορφή, μπορεί να δημιουργηθεί με μια κλήση της μεθόδου `Db4oEmbedded.newConfiguration()`. Αξίζει να σημειωθεί πως εάν το όνομα της βάσης δεδομένων που δόθηκε δεν υπάρχει, θα δημιουργηθεί. Επίσης, οι ρυθμίσεις με τις οποίες ξεκινάμε μια συνεδρία για μια συγκεκριμένη βάση δεδομένων πρέπει να είναι ίδιες, για λόγους συνέπειας και ασφάλειας. Σε περίπτωση που προσπαθήσουμε να χρησιμοποιήσουμε άλλες ρυθμίσεις, η συνεδρία δεν θα μπορέσει να ξεκινήσει.

Με την μέθοδο `close()` κλείνει η συνεδρία με την βάση δεδομένων και απελευθερώνονται όλοι οι αντίστοιχοι πόροι.

Αποθήκευση, Ενημέρωση Και Διαγραφή

Επόμενο βήμα είναι η διαδικασία της αποθήκευσης καινούργιων αντικείμενων, η τροποποίηση των ήδη αποθηκευμένων αντικείμενων και η διαγραφή τους. Η επεξήγηση θα βασιστεί πάνω στον παρακάτω κώδικα.

```
import com.db4o.*;

public class Example1 {

	final static String DB4OFILENAME = "formula1.db4o";

	public static void main(String[] args) {
		//Δημιουργούμε μια σύνδεση με την βάση δεδομένων
		ObjectContainer db = Db4oEmbedded.openFile(
				Db4oEmbedded.newConfiguration(), DB4OFILENAME);

		try {
			//Πραγματοποιούμε μερικές ενέργειες
			storeFirstPilot(db);
			storeSecondPilot(db);
			retrieveAllPilots(db);
			retrievePilotByName(db);
			updatePilot(db);
			deleteFirstPilotByName(db);
		} finally {
			//Κλείνουμε την σύνδεση
			db.close();
		}
	}
```

Σχήμα 14 Παράδειγμα Αποθήκευσης, Επεξεργασίας και Διαγραφής - μέρος 1ο

```
public static void storeFirstPilot(ObjectContainer db) {
	//Δημιουργούμε ένα καινούργιο αντικείμενο
	Pilot pilot1 = new Pilot("Michael Schumacher", 100);

	//Αποθηκεύουμε το αντικείμενο στη βάση
	db.store(pilot1);
}

public static void storeSecondPilot(ObjectContainer db) {
	//Δημιουργούμε ένα καινούργιο αντικείμενο
	Pilot pilot2 = new Pilot("Rubens Barrichello", 99);

	//Αποθηκεύουμε το αντικείμενο στη βάση
	db.store(pilot2);
}

public static void retrieveAllPilots(ObjectContainer db) {
	//Κάνουμε ένα ερώτημα στην βάση με βάση την κλάση που θέλουμε
	ObjectSet<Pilot> result = db.queryByExample(Pilot.class);

	//Προβολή αποτελεσμάτων
	//...
}

public static void retrievePilotByName(ObjectContainer db) {
	//Δημιουργούμε ένα αντικείμενο δείγνα
	Pilot proto = new Pilot("Michael Schumacher", 0);

	//Κάνουμε ερώτημα στην βάση με βάση το αντικείμενο δείγμα
	ObjectSet<Pilot> result = db.queryByExample(proto);

	//Προβολή αποτελεσμάτων
	//...
}

public static void updatePilot(ObjectContainer db) {
	//Κάνουμε ερώτημα στην βάση με βάση το αντικείμενο δείγμα
	ObjectSet<Pilot> result = db.queryByExample(
			new Pilot("Michael Schumacher", 0));

	//Ανακτούμε το πρώτο αποτέλεσμα
	Pilot found = result.next();

	//Τροποποιούμε το αντικείμενο μας
	found.addPoints(11);

	//Αποθηκεύουμε το αντικείμενο στη βάση
	db.store(found);

	//Ανακτούμε όλα τα αντικείμενα για να επιβεβαιώσουμε την
	//τροποποίηση
	retrieveAllPilots(db);
}
```

Σχήμα 15 Παράδειγμα Αποθήκευσης, Επεξεργασίας και Διαγραφής - μέρος 2ο

```
        public static void deleteFirstPilotByName(ObjectContainer db) {
                //Κάνουμε ερώτημα στην βάση με βάση το αντικείμενο δείγμα
                ObjectSet<Pilot> result = db.queryByExample(
                                new Pilot("Michael Schumacher", 0));

                //Ανακτούμε το πρώτο αποτέλεσμα
                Pilot found = result.next();

                //Διαγράφουμε το αντικείμενο
                db.delete(found);

                //Ανακτούμε όλα τα αντικείμενα για να επιβεβαιώσουμε την
                //τροποποίηση
                retrieveAllPilots(db);
        }
}
```

Σχήμα 16 Παράδειγμα Αποθήκευσης, Επεξεργασίας και Διαγραφής - μέρος 3ο

Έχοντας συνδεθεί στη βάση μπορούν τώρα να πραγματοποιηθούν οι επιθυμητές ενέργειες. Οι ενέργειες που χρησιμοποιήσαμε στο παράδειγμά μας είναι :

- Η δημιουργία δυο αντικειμένων και αποθήκευση στην βάση. Καλώντας τη μέθοδο `store()` και δίνοντας σαν όρισμα ένα αντικείμενο, το αντικείμενο αυτό αποθηκεύεται στην βάση.
- Ανάκτηση όλων των αντικειμένων τύπου Pilot. Στο συγκεκριμένο παράδειγμα χρησιμοποιούνται ο μηχανισμός ερωτήματος μέσω παραδείγματος. Ο μηχανισμός ερωτημάτων της db4o θα αναφερθεί παρακάτω.
- Ανάκτηση με βάση το όνομα.
- Επεξεργασία αντικειμένου και αποθήκευση του τροποποιημένου αντικειμένου. Ο `ObjectContainer`, όπως προαναφέρθηκε, παρακολουθεί τη μνήμη του προγράμματος. Όταν του ζητηθεί να αποθηκεύσει ένα αντικείμενο, βλέπει ότι το αντικείμενο αντιστοιχεί (καταλαμβάνει την ίδια περιοχή της μνήμης) με ένα αντικείμενο που ανακτήθηκε νωρίτερα από τη βάση δεδομένων. Οπότε, αντί να αποθηκευτεί ένα τρίτο αντικείμενο στην βάση δεδομένων, η μέθοδος `store()` ανανεώνει τα αποθηκευμένα δεδομένα με αυτά που υπάρχουν εκείνη τη στιγμή στη μνήμη.
- Διαγραφή αντικειμένου. Ισχύουν όσα προαναφέρθηκαν και για την `store()`. Εδώ θα διαγραφεί ένα αντικείμενο που ανακτήθηκε από τη βάση δεδομένων αμέσως προηγουμένως. Αν εκχωρούσαμε σαν όρισμα ένα αντικείμενο που δεν αντιστοιχείται, με βάση αυτά που προαναφέρθηκαν, τότε δε θα διαγραφόταν τίποτα.

Όταν τελειώσουμε τις δουλειές μας πρέπει να μεριμνήσουμε να κλείσουμε την σύνδεση με την βάση. Αν κλείναμε την σύνδεση στο ενδιάμεσο της εκτέλεσης του παραδείγματος, οι ενέργειες πάνω στη βάση δεδομένων θα δημιουργούσαν εξαιρέσεις. Αν κλείναμε και ξανανοίγαμε τη συνεδρία πριν την αποθήκευση του τροποποιημένου αντικειμένου, η `store()` θα αποθήκευε ένα ακόμα αντικείμενο στη βάση δεδομένων.

Ο `ObjectContainer` έχει επίσης και τις μεθόδους `commit()` και `rollback()`, οι γνωστοί μηχανισμοί από τις σχεσιακές βάσεις δεδομένων. Η επιτυχή εκτέλεση της `commit()` είναι που κάνει την db4o να εγγυάται ότι τα δεδομένα είναι αποθηκευμένα. Όταν καλείται η `close()` καλείται και η `commit()`. Είναι συνετό σε κάθε εφαρμογή να συνδυάζεται η χρήση της `store()` με την `commit()` ή μετά από ένα σύνολο συναλλαγών.

Ανάκτηση Δεδομένων

Η δυνατότητα ανάκτησης δεδομένων επιλεκτικά από επερώτηση (querying) αποτελεί βασικό χαρακτηριστικό των βάσεων δεδομένων.

Η db4o παρουσιάζει τρείς εναλλακτικές μεθόδους ερωτημάτων:

- Ερωτήματα με την χρήση πρότυπου (QueryByExample- QBE)
- Φυσικά ερωτήματα (Native Queries- NQ)
- SODA (Simple Object Data Access) ερωτήματα

Κάθε μία μέθοδος με τα πλεονεκτήματα και τα μειονεκτήματά της χρησιμοποιείται κατά περίπτωση. Αναλυτικότερα:

2.4 Ερωτήματα με την χρήση πρότυπου (QueryByExample-QBE)

Αυτός ο μηχανισμός χρησιμοποιείται με την κλήση της μεθόδου `queryByExample()` σε ένα `ObjectContainer`. Στην μέθοδο εκχωρείται σαν όρισμα ένα αντικείμενο που αποτελεί πρότυπο για τα αντικείμενα που θα ανακτηθούν. Τα αντικείμενα που θα ανακτηθούν έχουν τις ίδιες τιμές στα πεδία τους με αυτές του προτύπου. Τα πεδία του προτύπου που έχουν την προεπιλεγμένη τιμή (είναι null ή 0 / false για πρωταρχικά πεδία) αγνοούνται στη σύγκριση.

Ο μηχανισμός αυτός παρότι προσφέρει ίσως έναν εύκολο τρόπο, παρουσιάζει αρκετά μειονεκτήματα:

- Η σύγκριση που κάνει είναι μόνο για ισότητα. Δεν μπορούν να ανακτηθούν αντικείμενα βάσει ενός επιθυμητού εύρους τιμών σε ένα πεδίο.
- Αν η επιθυμητή τιμή είναι μια η προεπιλεγμένη, τότε θα αγνοηθεί.
- Πρέπει ο κώδικας της κλάσης να επιτρέπει τις όποιες μεταβολές που χρειάζονται για τη δημιουργία ενός προτύπου. Αυτό σημαίνει ότι ο δομητής δεν πρέπει να αρχικοποιεί όλα τα πεδία, και οι μέθοδοι που

επιτρέπουν να ορίζονται όλα τα επιθυμητά πεδία, να επιτρέπουν και πρωταρχικές τιμές.

- Δεν υποστηρίζει σύνθετα ερωτήματα (AND, OR, NOT, κλπ.)

Για την αποφυγή των παραπάνω περιορισμών η db4o παρέχει τα Native ερωτήματα. Στο παράδειγμα του προηγούμενου υποκεφαλαίου χρησιμοποιήσαμε αυτόν τον μηχανισμό για ανάκτηση δεδομένων.

2.5 Φυσικά ερωτήματα (Native Queries)

Αυτός ο μηχανισμός προορίζεται για χρήση στην πλειοψηφία των περιπτώσεων. Βασίζεται 100% πάνω στην γλώσσα προγραμματισμού, για αυτό και λέγονται Φυσικά. Τα ερωτήματα γράφονται στη γλώσσα που χρησιμοποιείται και ακολουθούν τη σημασιολογία της. Ελέγχονται κατά την ώρα της μεταγλώττισης, αλλαγές στον κώδικα της κλάσης του αντικειμένου προς διερεύνηση επηρεάζουν το ερώτημα, οι συγκρίσεις που γίνονται είναι μεταξύ ίδιου τύπου αντικειμένων. Γενικά, τα φυσικά ερωτήματα είναι εντελώς αντικειμενοστρεφή.

Η λειτουργία τους είναι απλή. Για αρχή χρειάζεται μια κλάση που επεκτείνει την αφηρημένη κλάση `Predicate<T>` της db4o. Το όρισμα `T` ορίζει τι τύπου θα είναι τα αντικείμενα που θα επεξεργαστεί η db4o για το επερώτημα. Η κλάση αυτή περιέχει την μέθοδο `match()`. Παίρνει σαν όρισμα ένα αντικείμενο τύπου `T`. Η μέθοδος επιστρέφει `true` ή `false`, ανάλογα με το αν επιθυμούμε ή όχι να περιέχεται το αντικείμενο στα αποτελέσματα. Ο κώδικας που περιέχετε στην `match()` μπορεί να είναι ουσιαστικά οτιδήποτε που υποστηρίζεται από την γλώσσα προγραμματισμού. Αυτό σημαίνει ότι ισχύουν και οι περιορισμοί που περιέχει ο κώδικας της κλάσης του αντικειμένου, π.χ. δεν μπορεί να επιτευχθεί απευθείας πρόσβαση σε ιδιωτικά πεδία εκτός και αν έχουν μέθοδο πρόσβασης και αυτή να είναι προσβάσιμη. Μετά από τα παραπάνω, δημιουργούμε ένα αντικείμενο αυτής της κλάσης και το εκχωρούμε σαν όρισμα στην κλήση της μεθόδου `query()` ενός `ObjectContainer`. Αυτό που κάνει η db4o όταν καλέσουμε την μέθοδο είναι, πρώτον, παίρνει τα αντικείμενα που βρίσκονται στην βάση και είναι υλοποίηση ή υποκλάσεις του `T`. Δεύτερον, κάθε ένα από αυτά τα αντικείμενα τα περνάει σας όρισμα στην μέθοδο `match()`. Η `query()` επιστρέφει ένα αντικείμενο `ObjectSet<T>`, ουσιαστικά είναι μια λίστα με κόμβους τύπου `T`. Υπάρχει η δυνατότητα να εκχωρηθεί, σαν δεύτερη παράμετρος, στην κλήση της μεθόδου `query()` και ένα αντικείμενο τύπου `Comparator` που θα καθορίζει τη σειρά με την οποία θα διαταχτούν τα αντικείμενα στη λίστα που θα μας επιστρέψει το ερώτημα. Γενικά όταν ανακτώνται αντικείμενα από την db4o επιστρέφονται με τη σειρά που αποθηκεύτηκαν.

Τα φυσικά ερωτήματα με την απλότητά τους προσφέρουν τη δυνατότητα να γραφούν ερωτήματα με τον ίδιο τρόπο που θα γραφόταν και οποιοδήποτε κομμάτι κώδικα στην εφαρμογή. Και αυτό με πλήρη ασφάλεια, μιας και ελέγχεται κατά τη μεταγλώττιση και θα ακολουθήσουν οποιεσδήποτε μετατροπές στον κώδικα.

Μοναδικό μειονέκτημά τους αποτελεί η απόδοση, σε σχέση με τον τρίτο μηχανισμό ερωτημάτων. Η db4o όταν εκτελεί φυσικά ερωτήματα, και ερωτήματα με παράδειγμα, ο κώδικάς τους μετατρέπεται σε κώδικα του τρίτου μηχανισμού, τα SODA ερωτήματα. Αυτό της επιτρέπει να αξιοποιεί ευρετήρια και να μη χρειάζεται για κάθε ερώτημα να ανακτά το αντικείμενο στη μνήμη για να κάνει τους ελέγχους. Αυτό όμως δε συμβαίνει πάντα. Η db4o αναλύει τον κώδικα του ερωτήματος και αν δεν μπορεί να το μετατρέψει σε SODA ερώτημα, το εκτελεί με υψηλό κόστος σε μνήμη επειδή όλα τα υπό διερεύνηση αντικείμενα ανακτώνται πρώτα στην μνήμη. Αυτή η περίπτωση ίσως θα έπρεπε να ανησυχεί τον προγραμματιστή, ιδιαίτερα όταν τα αντικείμενα τα οποία αξιολογεί κάθε ερώτημα είναι πολλές χιλιάδες ή/και όταν η μνήμη είναι ιδιαίτερα περιορισμένη. Αυτές οι περιπτώσεις όμως, στην πλειονότητα των περιπτώσεων, δεν απασχολούν τους προγραμματιστές στην αρχή της ανάπτυξης. Ακόμα και αν φτάσουν στο σημείο αυτό κατά την ανάπτυξη, η db4o παρέχει μηχανισμούς για παρακολούθηση και βελτιστοποίηση.

Ακολουθεί ένα παράδειγμα πάνω στα Native Queries.

```
import com.db4o.*;
import com.db4o.query.*;
import java.util.List;

public class NQExample {

	final static String DB4OFILENAME = "formula1.db4o";

	public static void main(String[] args) {
		//Δημιουργούμε μια σύνδεση με την βάση δεδομένων
		ObjectContainer db = Db4oEmbedded.openFile(
				Db4oEmbedded.newConfiguration(), DB4OFILENAME);

		try {
			//Πραγματοποιούμε μερικές ενέργειες
			storePilots(db);
			retrieveComplexNQ(db);
			retrieveArbitraryCodeNQ(db);
		} finally {
			//Κλείνουμε την σύνδεση
			db.close();
		}
	}

	public static void storePilots(ObjectContainer db) {
		//Δημιουργούμε και αποθηκεύουμε μερικά αντικείμενα
		db.store(new Pilot("Michael Schumacher", 100));
		db.store(new Pilot("Rubens Barrichello", 99));
	}
```

Σχήμα 17 Παράδειγμα Native Queries - μέρος 1ο

```
	public static void retrieveComplexNQ(ObjectContainer db) {
		//Εκτελούμε ένα ερώτημα, οι παράμετροι του οποίου ορίζονται
		//επι τόπου με μια ανώνυμη εσωτερική υλοποίηση της Predicate
		List<Pilot> result = db.query(new Predicate<Pilot>() {
			public boolean match(Pilot pilot) {
				return pilot.getPoints() > 99
					&& pilot.getPoints() < 199
					|| pilot.getName().equals("Rubens Barrichello");
			}
		});

		//Προβολή αποτελεσμάτων
		//...
	}

	public static void retrieveArbitraryCodeNQ(ObjectContainer db) {
		//Εκτελούμε ένα ερώτημα, οι παράμετροι του οποίου ορίζονται
		//απο ένα αντικείμενο μια υλοποίησης της κλάσης Predicate
		List<Pilot> result = db.query(new PredicateImpl(new int[]{1, 100}));

		//Προβολή αποτελεσμάτων
		//...
	}

	//Η κλάση που περιέχει τις παραμέτρους ενός ερωτήματος
	private static class PredicateImpl extends Predicate<Pilot> {

		private final int[] points;

		public PredicateImpl(int[] points) {
			this.points = points;
		}

		public boolean match(Pilot pilot) {
			for (int point : points) {
				if (pilot.getPoints() == point) {
					return true;
				}
			}
			return pilot.getName().startsWith("Rubens");
		}
	}
}
```

Σχήμα 18 Παράδειγμα Native Queries - μέρος 2ο

2.6 SODA ερωτήματα

Τα SODA ερωτήματα δουλεύουν σε χαμηλότερο επίπεδο από τα Φυσικά. Η λογική τους είναι διαφορετική. Παρόλο που στην πλειοψηφία τους οι εφαρμογές χρησιμοποιούν Φυσικά ερωτήματα, μπορεί να υπάρχουν εφαρμογές όπου απαιτείται δυναμική δημιουργία ερωτημάτων και τότε γίνεται λόγος για τα SODA ερωτήματα.

Όταν δημιουργείται ένα SODA ερώτημα πρέπει να θεωρούνται τα δεδομένα ως ένα σύνολο από κόμβους που κάποιοι συσχετίζονται μεταξύ τους.

Αυτοί οι κόμβοι είναι ουσιαστικά τα χαρακτηριστικά που έχουν τα αντικείμενα και οι συσχετίσεις είναι αυτές που δημιουργήθηκαν από το μοντόλο. Το όνομα κάθε κόμβου είναι το όνομα της αντίστοιχης μεταβλητής. Παρακάτω φαίνεται ένα παράδειγμα για το πώς δημιουργούμε και εκτελούμε ένα SODA ερώτημα.

```
import com.db4o.*;
import com.db4o.query.Constraint;
import com.db4o.query.Query;

public class SODAExample {

	final static String DB4OFILENAME = "formula1.db4o";

	public static void main(String[] args) {
		//Δημιουργούμε μια σύνδεση με την βάση δεδομένων
		ObjectContainer db = Db4oEmbedded.openFile(
				Db4oEmbedded.newConfiguration(), DB4OFILENAME);

		try {
			//Πραγματοποιούμε μερικές ενέργειες
			storePilots(db);
			retrieveAllPilots(db);
			retrievePilotByExactPoints(db);
			retrieveByConjunction(db);
			retrieveByComparison(db);
		} finally {
			//Κλείνουμε την σύνδεση
			db.close();
		}
	}

	public static void storePilots(ObjectContainer db) {
		//Δημιουργούμε και αποθηκεύουμε μερικά αντικείμενα
		db.store(new Pilot("Michael Schumacher", 100));
		db.store(new Pilot("Rubens Barrichello", 99));
	}

	public static void retrieveAllPilots(ObjectContainer db) {
		//Δημιουργία ερωτήματος
		Query query = db.query();

		//Ορισμός παραμέτρων
		//Ορίζουμε ότι θέλουμε όλα τ' αντικείμενα τύπου Pilot
		query.constrain(Pilot.class);

		//Εκτέλεση ερωτήματος
		ObjectSet result = query.execute();

		//Προβολή αποτελεσμάτων
		//...
	}
```

Σχήμα 19 Παράδειγμα SODA ερωτήματος - μέρος 1ο

```
	public static void retrievePilotByExactPoints(ObjectContainer db) {
		//Δημιουργία ερωτήματος
		Query query = db.query();

		//Ορισμός παραμέτρων
		//Ορίζουμε ότι θέλουμε όλα τ' αντικείμενα τύπου Pilot
		query.constrain(Pilot.class);
		//όπου το πεδίο points είναι ίσο με 100
		query.descend("points").constrain(100);

		//Εκτέλεση ερωτήματος
		ObjectSet result = query.execute();

		//Προβολή αποτελεσμάτων
		//...
	}

	public static void retrieveByConjunction(ObjectContainer db) {
		//Δημιουργία ερωτήματος
		Query query = db.query();

		//Ορισμός παραμέτρων
		//Ορίζουμε ότι θέλουμε όλα τ' αντικείμενα τύπου Pilot
		query.constrain(Pilot.class);
		//όπου το πεδίο name είναι ίσο με Michael Schumacher
		Constraint constr
			= query.descend("name").constrain("Michael Schumacher");
		//και το πεδίο points είναι ίσο με 99
		query.descend("points").constrain(99).and(constr);

		//Εκτέλεση ερωτήματος
		ObjectSet result = query.execute();

		//Προβολή αποτελεσμάτων
		//...
	}

	public static void retrieveByComparison(ObjectContainer db) {
		//Δημιουργία ερωτήματος
		Query query = db.query();

		//Ορισμός παραμέτρων
		//Ορίζουμε ότι θέλουμε όλα τ' αντικείμενα τύπου Pilot
		query.constrain(Pilot.class);
		//όπου το πεδίο points είναι μεγαλύτερο από 99
		query.descend("points").constrain(99).greater();

		//Εκτέλεση ερωτήματος
		ObjectSet result = query.execute();

		//Προβολή αποτελεσμάτων
		//...
	}
}
```

Σχήμα 20 Παράδειγμα SODA ερωτήματος - μέρος 2ο

Η διαδικασία δημιουργίας και εκτέλεσης ενός ερωτήματος SODA έχει ως εξής. Στην αρχή δημιουργείται ένα αντικείμενο τύπου `Query` καλώντας την μέθοδο `query()` στον `ObjectContainer` που δημιουργήθηκε όταν συνδεθήκαμε με τη βάση. Στο επόμενο βήμα ορίζονται οι παράμετροι στο ερώτημα με την κλήση των αντίστοιχων μεθόδου της `Query`. Με κάθε κλήση των μεθόδων της `Query` δημιουργούμε μια ουρά από περιορισμούς. Γενικά χρησιμοποιούμε την `constraint()` για να επιβάλλουμε περιορισμούς και την `descend()` για να περιηγηθούμε στο γράφο των δεδομένων μας. Τέλος, εκτελείται το ερώτημα καλώντας την μέθοδο `execute()`. Παρατηρείται ότι η εκτέλεση του ερωτήματος επιστρέφει ένα αντικείμενο του τύπου `ObjectSet`, όπως και με τα Φυσικά ερωτήματα. Στην περίπτωση που δεν ορισθούν παράμετροι, το ερώτημα θα επιστρέψει όλα τα αντικείμενα που είναι αποθηκευμένα στη βάση.

Στο σημείο αυτό αξίζει να σημειωθεί ότι η χρήση SODA ερωτημάτων σε μία εφαρμογή είναι από ανύπαρκτη έως μεμονωμένη. Κι αυτό γιατί όταν χρησιμοποιούνται δεν μπορεί να γίνει έλεγχος αν τα ονόματα των πεδίων είναι σωστά, ούτε αν οι συγκρίσεις που γίνονται είναι ασφαλείς. Με αποτέλεσμα να μην υπάρχει κανένας έλεγχος δηλαδή όταν μεταγλωττίζεται ο κώδικας.

2.7 Ποια μέθοδος ερωτημάτων είναι καλύτερη:

- Τα Φυσικά ερωτήματα αποτελούν το κύριο interface για τη db4o, οπότε θα πρέπει να προτιμώνται.
- Με την τρέχουσα κατάσταση της βελτιστοποίησης ερωτήματος στην db4o μπορεί να υπάρχουν ερωτήματα που θα εκτελεστούν γρηγορότερα σε SODA. Η επιλογή των SODA ερωτημάτων μπορεί επίσης να είναι πιο βολική για την κατασκευή δυναμικών ερωτημάτων κατά το χρόνο εκτέλεσης.
- Τα ερωτήματα με τη χρήση πρότυπου είναι χρήσιμα για απλές περιπτώσεις, αλλά είναι περιορισμένα στη λειτουργικότητα.

Βέβαια μπορεί να γίνει συνδυασμός των παραπάνω στρατηγικών ανάλογα με τις εκάστοτε ανάγκες της εφαρμογής. Στην εφαρμογή που δημιουργήθηκε στα πλαίσια της παρούσας εργασίας χρησιμοποιήθηκαν κατά κύριο λόγο Φυσικά ερωτήματα και σε ελάχιστες περιπτώσεις SODA, για τους λόγους που ήδη αναφέρθηκαν.

2.7 Συναλλαγές (Transactions)

Οι συναλλαγές είναι μεταβλητές μονάδες εκτέλεσης προγραμμάτων που προσπελαύνουν και πιθανόν ενημερώνουν διάφορα δεδομένα. Η db4o λειτουργεί πάντα με συναλλαγές. Μια συναλλαγή στην db4o ορίζεται ως το σύνολο των μεθόδων που καλούνται μεταξύ δυο κλήσεων της μεθόδου commit(). Μόνο με την κλήση της commit() η db4o εγγυάται την επιτυχή μονιμοποίηση των αντικειμένων και της συνέπειας της βάσης δεδομένων.

2.9 ACID Ιδιότητες

Οι ACID ιδιότητες είναι από τις σημαντικότερες έννοιες στην θεωρία των βάσεων δεδομένων. Καθορίζουν τις απαιτήσεις για την αξιοπιστία και την ακεραιότητα των βάσεων δεδομένων. Αυτές είναι:

- **Ατομικότητα (A**tomicity**).** Δηλαδή, πρέπει να ακολουθούν έναν κανόνα "όλα ή τίποτα". Κάθε συναλλαγή είτε ολοκληρώθηκε επιτυχώς, είτε όχι, η κατάσταση της βάσης δεδομένων δεν αλλάζει καθόλου.
- **Συνέπεια (C**onsistency**).** Εξασφαλίζει ότι η βάση δεδομένων παραμένει πάντα σε μια σταθερή κατάσταση. Κάθε συναλλαγή πηγαίνει τη βάση δεδομένων από μια συνεπή κατάσταση στην επόμενη συνεπή κατάσταση.
- **Απομόνωση (I**solation**).** Απομόνωση σημαίνει ότι διαφορετικές λειτουργίες δεν μπορούν να έχουν πρόσβαση στα τροποποιημένα δεδομένα από άλλη συναλλαγή που δεν έχει ακόμη ολοκληρωθεί.
- **Ανθεκτικότητα (D**urability**).** Αυτό αναφέρεται μόνο στον πραγματικό στόχο του κάθε χώρου αποθήκευσης δεδομένων. Σημαίνει απλώς ότι τα δεδομένα πρέπει να αποθηκεύονται μόνιμα.

Η db4o πληροί τις ιδιότητες ACID. Κάθε αντικείμενο έχει τη δική του συναλλαγή. Κάθε συναλλαγή είναι μια μονάδα εργασίας και εξασφαλίζει τις ιδιότητες ACID. Αυτό σημαίνει, ότι μια συναλλαγή στην db4o είναι μια ατομική λειτουργία. Οπότε, είτε όλες οι αλλαγές των συναλλαγών db4o δεσμεύονται και γίνονται μόνιμες, είτε σε περίπτωση βλάβης ή επαναφοράς καμία κατάσταση δεν αλλάζει. Με αυτόν τον τρόπο, η βάση δεδομένων διατηρείται συνεπής, ακόμη και αν η εφαρμογή ή η βάση δεδομένων πάψει να ανταποκρίνεται.

Φυσικά, μπορεί να γίνει και αναίρεση μίας συναλλαγής (rollback), απλώς καλώντας τη μέθοδο `rollback()`. Όμως, αξίζει να σημειωθεί ότι όταν αναιρούνται οι αλλαγές που πραγματοποιήθηκαν, η db4o δεν αναιρεί τα αντικείμενα στη μνήμη. Για να επιβεβαιωθεί ότι τα αντικείμενα στη μνήμη έχουν την ίδια κατάσταση όπως και στη βάση δεδομένων, θα πρέπει να ανανεωθούν (κλήση μεθόδου `refresh()`).

2.10 Απομόνωση (Isolation)

Η Απομόνωση επιβάλλει κανόνες που διασφαλίζουν ότι οι συναλλαγές δεν αλληλεπιδρούν μεταξύ τους, ακόμη και αν αυτές εκτελούνται ταυτόχρονα. Σε περίπτωση που δύο ή περισσότερες συναλλαγές εκτελούνται ταυτόχρονα, θα πρέπει να εκτελούνται με τέτοιο τρόπο, ώστε η συναλλαγή Α δεν θα επηρεάζεται από τα ενδιάμεσα δεδομένα της συναλλαγής Β. Εάν το αποτέλεσμα στη βάση

δεδομένων είναι το ίδιο όταν οι συναλλαγές εκτελούνται ταυτόχρονα ή όταν η εκτέλεσή τους παρεμβάλλεται, οι πράξεις αυτές ονομάζονται σειριοποιήσιμες.
Επίπεδα απομόνωσης (isolation levels):

- **Σειριοποιήσιμο (Serializable):** Σε αυτή την περίπτωση, οι συναλλαγές εκτελούνται σειριακά, έτσι ώστε να μην υπάρχει ταυτόχρονη πρόσβαση δεδομένων. Οι συναλλαγές μπορούν να εκτελούνται ταυτόχρονα, αλλά μόνο όταν η ψευδαίσθηση των σειριακών συναλλαγών διατηρείται (δηλαδή δε συμβαίνει ταυτόχρονη πρόσβαση στα δεδομένα). Ωστόσο, αν το σύστημα ανιχνεύσει μια ταυτόχρονη συναλλαγή σε εξέλιξη η οποία παραβιάζει την ψευδαίσθηση σειριοποιησιμότητας, θα πρέπει να αναγκάσει τη συναλλαγή να αναιρεθεί, και η εφαρμογή θα πρέπει να επανεκκινήσει τη συναλλαγή.
- **Επαναλαμβανόμενη Ανάγνωση (Repeatable Read) :** Σε αυτό το επίπεδο απομόνωσης, όλες οι λειτουργίες ανάγνωσης σε μία συναλλαγή δίνουν το ίδιο αποτέλεσμα.
- **Δεσμευμένη Ανάγνωση (Read Committed) :** Στην περίπτωση αυτή, μόνο οι τιμές που έχουν τροποποιηθεί και δεσμεύτηκαν από άλλες συναλλαγές μπορούν να διαβαστούν. Αυτό είναι και το μοντέλο που χρησιμοποιεί η db4o.
- **Μη Δεσμευμένη Ανάγνωση (Read Uncommitted):** Σε αυτό το επίπεδο απομόνωσης, οι τιμές που έχουν τροποποιηθεί από άλλες συναλλαγές μπορούν να διαβαστούν πριν δεσμευτούν.

2.11 Σχέδιο ενεργοποίησης (Activation Concept)

Η ενεργοποίηση είναι ένας μηχανισμός της db4o που ελέγχει τη δημιουργία του αντικειμένου. Όταν ένα ερώτημα ανακτά αντικείμενα, τα πεδία τους φορτώνονται στη μνήμη, μόνο σε ένα ορισμένο βάθος ενεργοποίησης. Σε αυτή την περίπτωση με την έννοια βάθος νοείται ο «αριθμός των αναφορών των μελών μακριά από το αρχικό αντικείμενο». Όλα τα πεδία σε χαμηλότερα επίπεδα, κάτω από το βάθος ενεργοποίησης, καθορίζονται σε null ή στις προεπιλεγμένες τιμές. Έτσι λοιπόν, η db4o δε φορτώνει ολόκληρο το αντικείμενο του γραφήματος από τη βάση δεδομένων. Αντ' αυτού, φορτώνει μόνο τμήματα του αντικειμένου του γραφήματος που ενδιαφέρουν τον χρήστη.

Η ενεργοποίηση συμβαίνει στις εξής περιπτώσεις :

1. Όταν γίνεται επανάληψη σε αποτελέσματα του ερωτήματος
2. Όταν ένα αντικείμενο ενεργοποιηθεί ρητά μαζί με τη μέθοδο ενεργοποίησης των containers των αντικειμένων
3. Οι συλλογές των μελών ενεργοποιούνται αυτόματα, όταν ενεργοποιείται η συλλογή, χρησιμοποιώντας τουλάχιστον depth 1 για λίστες και depth 2 για χάρτες.

Σκοπός της ενεργοποίησης είναι να βελτιώσει την απόδοση και να εξοικονομήσει μνήμη. Ωστόσο, εάν το βάθος ενεργοποίησης ορισθεί πολύ υψηλά, θα υπάρξει μείωση της απόδοσης.

2.12 Διαφανής Ενεργοποίηση (Transparent Activation)

Στην περίπτωση που εξετάζεται ένα πολύ σύνθετο μοντέλο ή δεν χρειάζεται ιδιαίτερη ασχολία με όλα τα προβλήματα ενεργοποίησης, τότε η διαφανής ενεργοποίηση είναι η καλύτερη επιλογή. Η Διαφανής Ενεργοποίηση καθιστά δυνατή την ενεργοποίηση των αντικειμένων χωρίς τη βοήθεια του προγραμματιστή.

Όταν η db4o εκτελείται σε κατάσταση Διαφανούς ενεργοποίησης δεν υπάρχουν εκπλήξεις με null μέλη που δεν έχουν ακόμη διαβαστεί από τη βάση δεδομένων. Παρά το γεγονός ότι δεν υπάρχουν εκπλήξεις με τα null αντικείμενα όταν εργαζόμαστε με αυτόν τον τρόπο, η πρόσβαση στη ρίζα ενός γραφήματος μεγάλου βάθους, μπορεί να φορτώσει ολόκληρο το γράφημα στη μνήμη και μπορεί να σπαταλήσει πολύ χρόνο και να χρησιμοποιήσει πολλή μνήμη.

Κεφάλαιο 3 - Η χρήση της db4o στην εφαρμογή

Εισαγωγή

Αφού έγινε η επιτυχής εγκατάσταση της db4o, σύμφωνα με τις οδηγίες που προαναφέρθηκαν, ξεκίνησε η δόμηση της `enum DB`. Η `enum DB` αποσκοπεί στο να παρέχει στην εφαρμογή μοναδική πρόσβαση στην βάση δεδομένων. Έχουμε δημιουργήσει μια αφηρημένη κλάση `Entity` που είναι η βάση για κάθε αντικείμενο που θέλουμε να αποθηκεύσουμε στην βάση (περιορισμός που ορίζεται απ' την υλοποίηση και όχι απ' την db4o). Αυτό έγινε επειδή θέλαμε κάθε οντότητα, πρώτον, να υλοποιεί την `equals()`, και δεύτερον, να υλοποιεί την διεπαφή `Comparable`. Επίσης έχουμε ορίσει την αφηρημένη κλάση `Property`, που είναι υποκλάση της `Entity`, για να δηλώσουμε ιδιότητες για τις οντότητές μας. Αυτό έγινε με το σκεπτικό, όταν θέλουμε να προσθέσουμε μια ιδιότητα σε μια οντότητα, πχ αποφασίσαμε ότι θέλουμε να αποθηκεύουμε και το τηλέφωνο ενός ατόμου, να μην χρειάζεται να μεταβάλουμε την κλάση που υπάρχει αλλά απλά να δημιουργήσουμε μια νέα Ιδιότητα. Τα υποκεφάλαια 3.1 μέχρι 3.7 παρουσιάζουν την υλοποίηση της `enum DB`. Στο υποκεφάλαιο 3.8 παρουσιάζουμε ενδεικτικά τις υλοποιήσεις των οντοτήτων. Τα τελευταία υποκεφάλαια παρουσιάζουν την χρήση της `enum DB` μέσα στην εφαρμογή.

3.1 Πρόσβαση στη βάση

```
Instance;
private ObjectContainer db;
private final static String FILENAME = "_db\\DB.db4o";

private DB() {
	db = Db4oEmbedded.openFile(
			Db4oEmbedded.newConfiguration(), FILENAME);

	EventRegistry eventRegistry
			= EventRegistryFactory.forObjectContainer(db);
	eventRegistry.creating().addListener(new EventCreating());
	eventRegistry.deleted().addListener(new EventDeleted());
}
```

Σχήμα 21 enum DB - Δημιουργία της σύνδεσης

Δημιουργήθηκε μία μεταβλητή `db` τύπου `ObjectContainer` (όπου `ObjectContainer` το interface για την πρόσβαση στη βάση δεδομένων) και μία `FILENAME` τύπου `String`. Η εντολή `Db4oEmbedded.openFile()`, όπως έχει

ήδη αναφερθεί, περιέχει ως παραμέτρους την `Db4oEmbedded.newConfiguration()` και τη διαδρομή προς το αρχείο της βάσης δεδομένων.

Τα αντικείμενα τύπου `EventRegistry` χρησιμοποιούνται για να παρακολουθούμε τα γεγονότα που συμβαίνουν στην βάση δεδομένων. Δημιουργήσαμε μια κλάση, την `EventCreating`, για να ελέγχει αν ένα αντικείμενο υπάρχει στη βάση. Αυτός ο έλεγχος γίνεται κάθε φορά που η db4o προσπαθεί να αποθηκεύσει ένα καινούργιο αντικείμενο στην βάση δεδομένων. Αντίστοιχα δημιουργήσαμε την `EventDeleted` για το γεγονός της διαγραφής.

3.2 Commit και rollback

Μπορεί να γίνει commit στη συναλλαγή ανά πάσα στη στιγμή και να αποθηκευτούν τυχόν αλλαγές. Ωστόσο, σε περίπτωση που η `commit()` «πετάξει» exceptions (δηλαδή αδυναμία ολοκλήρωσης του commit), καλείται η `rollback()` για να επανέρθουμε στην κατάσταση στο τελευταίο commit.

```
private void commit() {
	try {
		db.commit();
	} catch (Exception e) {
		db.rollback();
	}
}
```

Σχήμα 22 enum DB - μέθοδος commit()

3.3 Κλείσιμο της σύνδεσης

Καλώντας τη μέθοδο close(), κλείνει το αρχείο της βάσης και απελευθερώσει όλους τους πόρους που σχετίζονται με αυτή.

```
public void close() {
	db.close();
}
```

Σχήμα 23 enum DB - μέθοδος close()

3.4 Αποθήκευση Αντικειμένων

Για την αποθήκευση ενός αντικειμένου, απλώς καλείται η μέθοδος

```
public <E extends Entity<E>> E store(E Entity) {
	db.store(Entity);
	commit();
	return Entity;
}
```

Σχήμα 24 enum DB - μέθοδος store()

`store()` στη βάση με όρισμα το αντικείμενο που πρόκειται να αποθηκευτεί. Έπειτα, καλείται η `commit()`, ώστε να διασφαλιστεί η αποθήκευση των δεδομένων. Όπως αναφέραμε στην αρχή, η μέθοδος δέχεται μόνο αντικείμενα που υλοποιούν την Entity.

3.5 Διαγραφή Αντικειμένων

Για τη διαγραφή ενός αντικειμένου, απλώς καλείται η μέθοδος `delete()` στη βάση με όρισμα το αντικείμενο που πρόκειται να διαγραφεί. Έπειτα, καλείται η `commit()`, ώστε να διασφαλιστεί η διαγραφή των δεδομένων.

```
public <E extends Entity<E>> E delete(E Entity) {
	db.delete(Entity);
	commit();
	return null;
}
```

Σχήμα 25 enum DB - μέθοδος delete()

3.6 Ανάκτηση Αντικειμένων

Βασιστήκαμε σχεδόν αποκλειστικά στα Native Queries για την ανάκτηση αντικειμένων. Επιλέχθηκε η συγκεκριμένη μέθοδος ερωτημάτων επειδή λειτουργεί τόσο απλά και αποτελεσματικά. Η μόνη εξαίρεση είναι όταν θέλουμε απλά να ανακτήσουμε αντικείμενα συγκεκριμένου τύπου, όπου η ανάκτηση γίνεται με SODA ερώτημα.

Η ανάκτηση γίνεται με την μέθοδο `query()` όπου έχουμε τρεις υπερφορτώσεις της μεθόδου. Η πρώτη παίρνει σαν όρισμα ένα αντικείμενο τύπου `Class` όπου την χρησιμοποιούμε για την ανάκτηση συγκεκριμένου τύπου αντικειμένων, όπου ο τύπος ορίζεται απ' το όρισμα. Η υλοποίηση βασίζεται σε SODA ερωτήματα. Η δεύτερη υπερφόρτωση παίρνει σαν όρισμα ένα αντικείμενο τύπου `Predicate` και το αν θέλουμε να γίνει ταξινόμηση ή όχι. Η υλοποίηση βασίστηκε σε Native Queries. Τέλος, η τρίτη υπερφόρτωση είναι σαν την δεύτερη

όπου αντί να ορίζουμε αν θέλουμε ή όχι ταξινόμηση, δίνουμε ένα αντικείμενο

```
public <E extends Entity<E>> List<E> query(Class<E> EntityType) {
	Query q = db.query();

	q.constrain(EntityType);

	return q.execute();
}

public <E extends Entity<E>> List<E> query(Predicate<E> predicate,
                                           boolean sorted) {
	if (sorted) {
		return db.query(predicate, new QComparator<E>());
	}
	return db.query(predicate);
}

public <E extends Entity<E>> List<E> query(Predicate<E> predicate,
                                           Comparator<E> comparator) {
	return db.query(predicate, comparator);
}
```

Σχήμα 26 enum DB - οι βασικές μέθοδοι για query

τύπου `Comparator` όπου ορίζουμε πως θέλουμε να γίνει ταξινόμηση.

3.7 Βοηθητικές μέθοδοι και κλάσεις

Τα παραπάνω ήταν ο βασικός κώδικας της `enum DB`. Παρακάτω δημιουργήσαμε τις λεγόμενες βοηθητικές μεθόδους όπου απλά γενικεύσαμε κάποιες περιπτώσεις χρήσης των ερωτημάτων στην εφαρμογή μας όσο έχει να κάνει με την ανάκτηση βάσει των ιδιοτήτων ή την ειδική ανάκτηση ιδιοτήτων.

```
public <P extends Property<P, ?, E>,
        E extends Entity<E>> List<P> q(Class<P> classType,
                                       E entity,
                                       boolean sorted) {
	if (sorted) {
		return db.query(new PropertyByEntity<P, E>(classType,
                                                           entity),
                                new QComparator<P>());
	}
	return db.query(new PropertyByEntity<P, E>(classType, entity));
}

public <P extends Property<P, V, ?>,
        V extends Comparable<V>> List<P> qValue(Class<P> classType,
                                                V value,
                                                boolean sorted) {
	if (sorted) {
		return db.query(new PropertyByValue<P, V>(classType,
                                                          value),
                                new QComparator<P>());
	}
	return db.query(new PropertyByValue<P, V>(classType, value));
}
```

Σχήμα 27 enum DB - βοηθητικές μέθοδοι ερωτημάτων - μέρος 1ο

```
public <P extends Property<P, ?, X>,
        X extends Property<X, ?, E>,
        E extends Entity<E>> List<P> qDeep(Class<P> classType,
                                           E entity,
                                           boolean sorted) {
       if (sorted) {
              return db.query(new PropertyByDeepEntity<P, X, E>(classType,
                                                                 entity),
                              new QComparator<P>());
       }
       return db.query(new PropertyByDeepEntity<P, X, E>(classType,
                                                          entity));
}

public <P extends Property<P, V, E>,
        V extends Comparable<V>,
        E extends Entity<E>> List<P> q(Class<P> classType,
                                       E entity,
                                       V value,
                                       boolean sorted) {
       if (sorted) {
              return db.query(new PropertyByEntityAndValue<P, V, E>(classType,
                                                                    entity,
                                                                    value),
                              new QComparator<P>());
       }
       return db.query(new PropertyByEntityAndValue<P, V, E>(classType,
                                                             entity,
                                                             value));
}

public <P extends Property<P, V, E>,
        V extends Comparable<V>,
        E extends Entity<E>> List<E> q2(Class<E> classType,
                                        V value,
                                        boolean sorted) {
       if (sorted) {
              return db.query(new EntityByPropertysValue<P, V, E>(classType,
                                                                  value),
                              new QComparator<E>());
       }
       return db.query(new EntityByPropertysValue<P, V, E>(classType, value));
}

public <P extends Property<P, ?, X>,
        X extends Entity<X>,
        E extends Entity<E>> List<E> q(P property, boolean sorted) {
       if (sorted) {
              return db.query(new EntityByPropertysEntity<P, X, E>(property),
                              new QComparator<E>());
       }
       return db.query(new EntityByPropertysEntity<P, X, E>(property));
}
```

Σχήμα 28 enum DB - βοηθητικές μέθοδοι ερωτημάτων - μέρος 2ο

```
public <P extends Property<P, ?, X>,
        X extends Entity<X>,
        E extends Entity<E>> List<P> qValue(Class<P> classType,
                                            E entity,
                                            boolean sorted) {
        if (sorted) {
                return db.query(new PropertyByDeepValue<P, X, E>(classType,
                                                                 entity),
                                new QComparator<P>());
        }
        return db.query(new PropertyByDeepValue<P, X, E>(classType,
                                                         entity));
}
```

Σχήμα 29 enum DB - βοηθητικές μέθοδοι ερωτημάτων - μέρος 3ο

Έπειτα ακολουθεί μία κλάση σύγκρισης ερωτημάτων, που την χρησιμοποιούμε στην προεπιλεγμένη περίπτωση που θέλουμε ταξινόμηση των αποτελεσμάτων:

```
private static class QComparator<V extends Comparable<V>>
                     implements QueryComparator<V> {

        @Override
        public int compare(V target1, V target2) {
                if (target1 == null) {
                        return -1;
                }
                return target1.compareTo(target2);
        }
}
```

Σχήμα 30 enum DB - βοηθητική κλάση ταξινόμησης

Τέλος, ορίζουμε μερικές βοηθητικές κλάσεις για διάφορες βοηθητικές υλοποιήσεις της Predicate.

```
private static class PropertyByEntity<P extends Property<P, ?, E>,
                                      E extends Entity<E>>
                                     extends Predicate<P> {
        private final E entity;

        public PropertyByEntity(Class<P> classType, E entity) {
                super(classType);
                this.entity = entity;
        }

        @Override
        public boolean match(P candidate) {
                return candidate.getEntity().equals(entity);
        }
}
```

Σχήμα 31 enum DB - βοηθητικές κλάσεις υλοποίησης της Predicate - μέρος 1ο

```
private static class PropertyByValue<P extends Property<P, V, ?>,
                                     V extends Comparable<V>>
                                     extends Predicate<P> {
	private final V value;

	public PropertyByValue(Class<P> classType, V value) {
		super(classType);
		this.value = value;
	}

	@Override
	public boolean match(P candidate) {
		return candidate.getValue().equals(value);
	}
}

private static class PropertyByDeepEntity<P extends Property<P, ?, X>,
                                          X extends Property<X, ?, E>,
                                          E extends Entity<E>>
                                          extends Predicate<P> {
	private final E entity;

	public PropertyByDeepEntity(Class<P> classType, E entity) {
		super(classType);
		this.entity = entity;
	}

	@Override
	public boolean match(P candidate) {
		return candidate.getEntity().getEntity().equals(entity);
	}
}

private static class PropertyByEntityAndValue<P extends Property<P, V, E>,
                                              V extends Comparable<V>,
                                              E extends Entity<E>>
                                              extends Predicate<P> {
	private final E entity;
	private final V value;

	public PropertyByEntityAndValue(Class<P> classType,
	                                E entity,
	                                V value) {
		super(classType);
		this.entity = entity;
		this.value = value;
	}

	@Override
	public boolean match(P candidate) {
		return candidate.getEntity().equals(entity)
		       && candidate.getValue().equals(value);
	}
}
```

Σχήμα 32 enum DB - βοηθητικές κλάσεις υλοποίησης της Predicate - μέρος 2ο

```
private static class PropertyByDeepValue<P extends Property<P, ?, X>,
                                         X extends Entity<X>,
                                         E extends Entity<E>>
                                         extends Predicate<P> {
	private final E entity;

	public PropertyByDeepValue(Class<P> type, E r) {
		super(type);
		entity = r;
	}

	@Override
	public boolean match(P candidate) {
		@SuppressWarnings("unchecked")
		List<X> q = DB.Instance.q(Property.class,
					candidate.getEntity(),
					entity,
					false);
		return q != null && !q.isEmpty();
	}
}

private static class EntityByPropertysEntity<P extends Property<P, ?, X>,
                                             X extends Entity<X>,
                                             E extends Entity<E>>
                                             extends Predicate<E> {
	private final P property;

	public EntityByPropertysEntity(P property) {
		super();
		this.property = property;
	}

	@Override
	public boolean match(E candidate) {
		@SuppressWarnings("unchecked")
		List<X> q = DB.Instance.q(Property.class,
					property.getEntity(),
					candidate,
					false);
		return q != null && !q.isEmpty();
	}
}
```

Σχήμα 33 enum DB - βοηθητικές κλάσεις υλοποίησης της Predicate - μέρος 3ο

```
	private static class EntityByPropertysValue<P extends Property<P, V, E>,
	                                            V extends Comparable<V>,
	                                            E extends Entity<E>>
	                                            extends Predicate<E> {
		private final V value;

		public EntityByPropertysValue(Class<E> type, V value) {
			super(type);
			this.value = value;
		}

		@Override
		public boolean match(E candidate) {
			@SuppressWarnings("unchecked")
			List<P> q = DB.Instance.q(Property.class,
			                          candidate,
			                          value,
			                          false);
			return q != null && !q.isEmpty();
		}
	}
```

Σχήμα 34 enum DB - βοηθητικές κλάσεις υλοποίησης της Predicate - μέρος 4ο

3.8 Υλοποίηση οντοτήτων

Η παρούσα εργασία, όπως έχει ήδη αναφερθεί, αποτελείται από τρεις οντότητες σε κάθε μία από τις οποίες αντιστοιχεί και μία κλάση:

- Μάθημα (Course.java)
- Καθηγητής (Professor.java)
- Αίθουσα Διδασκαλίας (Room.java)

Η δομή των παραπάνω κλάσεων είναι κοινή, γι' αυτό και ενδεικτικά θα αναλυθεί ο κώδικας της κλάσης Course.java.

```
public class Course extends Entity<Course> {
	private String Title = "";
	private Integer Semester = 0;
	private String Simple = "";

	public Course() {}

	public String getTitle() {return Title;}

	public Integer getSemester() {return Semester;}

	public String getSimple() {return Simple;}

	public void setTitle(String title) {this.Title = title;}

	public void setSemester(Integer semester) {this.Semester = semester;}

	public void setSimple(String Simple) {this.Simple = Simple;}
```

Σχήμα 35 κλάση Cource - μέρος 1ο

Αρχικά, ορίζονται και αρχικοποιούνται οι τρείς απαραίτητες μεταβλητές: Τίτλος Μαθήματος (`Title`), Τυπικό Εξάμηνο Μαθήματος (`Semester`) και Συντομογραφία Μαθήματος (`Simple`). Έπειτα, αφού ορισθεί ο μη παραμετρικός δομητής `Course()` υλοποιούνται οι αντίστοιχοι μέθοδοι getters και setters.
Ακολουθεί η μέθοδος `compareTo()`, η οποία λειτουργεί μόνο αν η παράμετρος είναι ορισμένη, συγκρίνει αν το εξάμηνο είναι ίδιο με το εξάμηνο του μαθήματος και αν ναι συγκρίνει τον τίτλο με τον τίτλο του μαθήματος:

```
@Override
public int compareTo(Course o) {
	if (o == null) {return 1;}

	int result = Semester.compareTo(o.Semester);

	if (result == 0) {result = Title.compareTo(o.Title);}

	return result;
}
```

Σχήμα 36 κλάση Cource - μέρος 2ο - μέθοδος compareTo()

Στη συνέχεια έχουμε τη μέθοδο `equals()` η οποία δέχεται σαν όρισμα ένα αντικείμενο `obj` και ελέγχει τους τίτλους και τα εξάμηνα των μαθημάτων. Ακολουθείται από την `hashCode()`. Στην Java οι δυο υλοποιήσεις πάνε μαζί επειδή μερικές φορές η JVM μπορεί να μην χρησιμοποιήσει την `equals()` για την σύγκριση αλλά την `hashcode()`.

```
@Override
public boolean equals(Object obj) {
	if (obj == null) {return false;}
	if (getClass() != obj.getClass()) {return false;}
	final Course other = (Course) obj;
	if ((this.Title == null)
		? (other.Title != null)
		: !this.Title.equals(other.Title)) {
		return false;
	}
	if (this.Semester != other.Semester
		&& (this.Semester == null
		|| !this.Semester.equals(other.Semester))) {
		return false;
	}
	return true;
}

@Override
public int hashCode() {
	int hash = 7;
	hash = 59 * hash + (this.Title != null
				? this.Title.hashCode() : 0);
	hash = 59 * hash + (this.Semester != null
				? this.Semester.hashCode() : 0);
	return hash;
}
```

Σχήμα 37 κλάση Cource - μέρος 3ο - μέθοδοι equals() και hashcode()

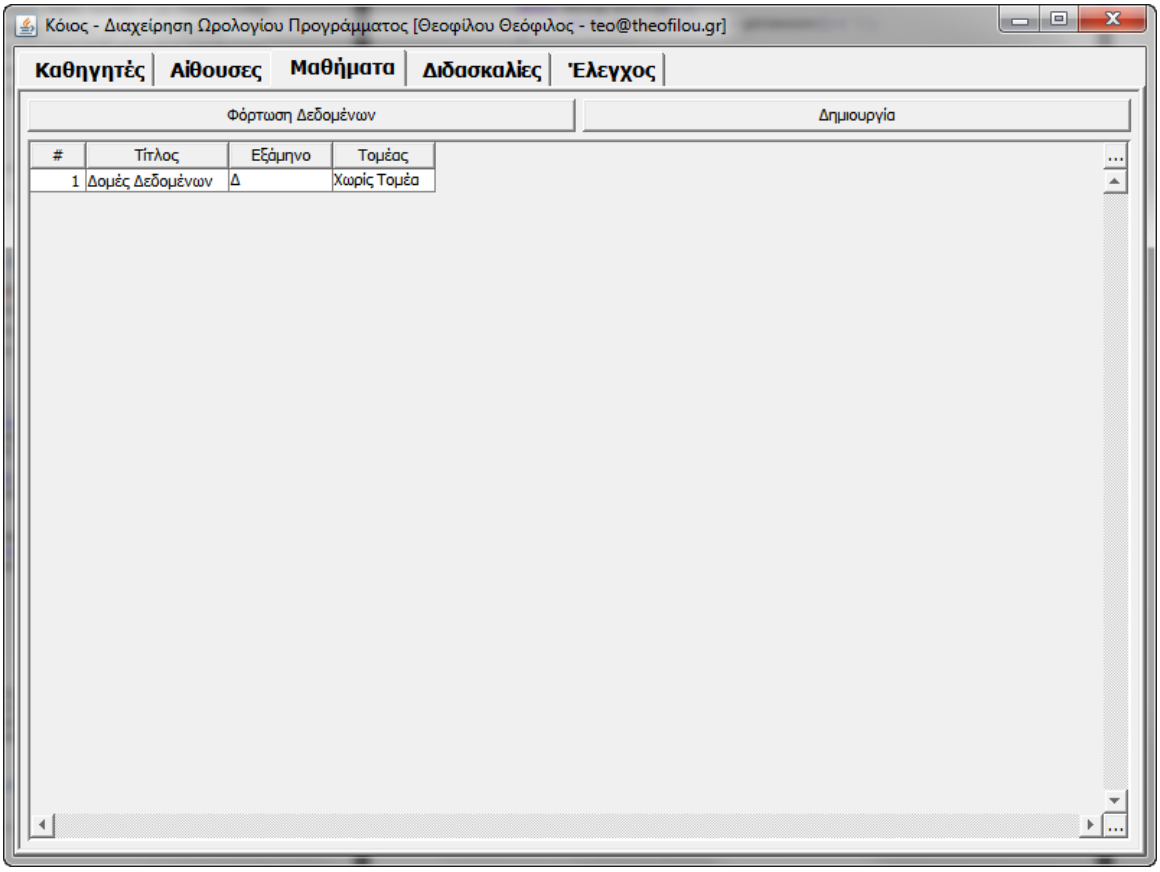
Και τέλος, η `toString()`:

```
@Override
public String toString() {
	return getTitle() + " [" + getSemester() + "]";
}
```

Σχήμα 38 κλάση Cource - μέρος 4ο - μέθοδος toString()

3.9 Οι βασικές λειτουργίες της db4o μέσα απ' την εφαρμογή

Οι βασικές λειτουργίες της db4o, αποθήκευση, ανάκτηση και διαγραφή, υλοποιούνται με ανάλογο τρόπο σε όλες τις καρτέλες που έχουν εφαρμογή. Οπότε δεν θα περιγράψουμε αναλυτικά τις λειτουργίες αυτές σε κάθε τους εμφάνιση αλλά μέσω της καρτέλας Μαθήματα. Σ' αυτήν υπάρχουν οι πιο περίπλοκες, ας πούμε, υλοποιήσεις αυτών των λειτουργιών. Για αρχή, ας θυμηθούμε την καρτέλα Μαθήματα.



Σχήμα 39 Η καρτέλα Μαθήματα

Η μόνη λειτουργία εδώ, που έχει σχέση με την βάση δεδομένων, είναι η Φόρτωση Δεδομένων. Όταν κληθεί η εφαρμογή να εκτελέσει αυτήν την εντολή, ουσιαστικά εκτελεί τον παρακάτω κώδικα.

```
table.clear();
for (Course course : DB.Instance.query(Course.class)) {
	@SuppressWarnings("unchecked")
	List<Sector> q = DB.Instance.q(Sector.class, course, false);
	table.addRow(course,
			null,
			course.getTitle(),
			course.getSemester(),
			q != null && !q.isEmpty()
			 ? q.get(0).getSector() : null);
}
```

Σχήμα 40 Ο κώδικας Φόρτωση Δεδομένων

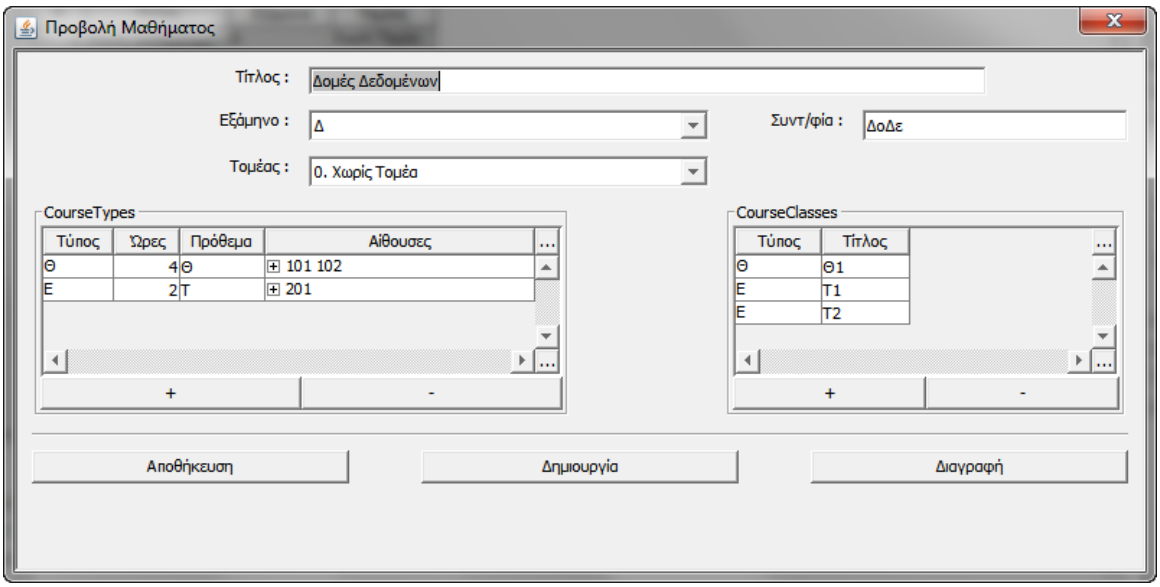
Ο πίνακας που εμφανίζει τα αποθηκευμένα αντικείμενα έχει τις εξής στήλες:

- # ο αύξων αριθμός της γραμμής του πίνακα
- Τίτλος ο τίτλος του, χαρακτηριστικό της κλάσης
- Εξάμηνο σε ποιο εξάμηνο διδάσκεται, χαρακτηριστικό της κλάσης
- Τομέας ο τομέας στον οποίο ανήκει το μάθημα, ιδιότητα

Οπότε, τα βήματα που εκτελεί ο κώδικας είναι:

A. Καθαρισμός του πίνακα
   [`table.clear()`]
B. Ανάκτηση όλων των Μαθημάτων
   [`DB.Instance.query(Course.class)`]
C. Για κάθε μάθημα
   [ο βρόγχος `for`]
   I. Ανάκτηση όλων των ιδιοτήτων τύπου Sector (ο Τομέας)
      [`DB.Instance.q(Sector.class, course, false)`]
   II. Προσθήκη γραμμής στον πίνακα δίνοντας τις τιμές στις αντίστοιχες στήλες, οι τιμές των παραμέτρων αντιστοιχούν:
      1. Το αντικείμενο που αντιστοιχεί στην γραμμή
      2. Στήλη αρίθμησης
      3. Ο Τίτλος του Μαθήματος
      4. Το εξάμηνο του Μαθήματος
      5. Ο Τομέας του, null αν δεν έχει οριστεί

Όταν θελήσουμε να επεξεργαστούμε ή να δημιουργήσουμε ένα καινούργιο αντικείμενο, δεν χρειάζεται επιπλέον αλληλεπίδραση με την βάση επειδή το αντικείμενο υπάρχει ήδη. Οπότε προχωράμε στην Προβολή Μαθήματος, που είναι ίδια είτε μιλάμε για επεξεργασία είτε για δημιουργία.



Σχήμα 41 Προβολή Μαθήματος

Όσο αφορά τα δεδομένα, υπάρχουν δυο κατηγορίες. Τα δεδομένα που έχουν να κάνουν με την κλάση Course (Τίτλος, Εξάμηνο, Συντ/φία) και τα δεδομένα που αφορούν Ιδιότητές της (Τομέας, CourseTypes) ή αντίστοιχες Ιδιότητές τους (CourseClass). Ο κώδικας που φορτώνει αυτά τα δεδομένα στην φόρμα είναι ο εξής :

```
view.Title.setData(course.getTitle());
view.Semester.setData(course.getSemester());
view.Simple.setData(course.getSimple());

for (ActionListener prop : read.values())
	if (prop != null)
		prop.actionPerformed(null);
```

Σχήμα 42 ο κώδικας που φορτώνει τα δεδομένα στην Προβολή Μαθήματος

Η μεταβλητή `view` είναι ένα `Panel` το οποίο χρησιμοποιούμε για να περιέχει τα στοιχεία (components) που χρειάζεται η φόρμα μας. Στην συγκεκριμένη περίπτωση, οι τρεις πρώτες γραμμές αποδίδουν εκεί που πρέπει τα δεδομένα από το αντικείμενο. Έπειτα γίνεται η απόδοση των τιμών από τις Ιδιότητες. Η διεπαφή χρήστη που έχουμε δημιουργήσει, έχει την λογική ότι κάθε στοιχείο που προστίθεται σε μια φόρμα για να παρουσιάζει τις τιμές μιας Ιδιότητας, πρέπει να δηλώνει στο `HashMap read` την μέθοδο ανάγνωσης που έχει. Οπότε με την `for` στο τέλος, ανακτώνται αυτά και καλούνται. Ενδεικτικά, μια από τις μεθόδους που καλούνται είναι αυτή της CourseTypes που βλέπουμε παρακάτω.

```
for (CourseType ct : CoursesActions.queryTypes()) {
	if (ct != null) {
		List<CourseTypeRoom> q = DB.Instance.q(CourseTypeRoom.class, ct, true);
		table.addRow(ct,
				ct.getType(),
				ct.getHours(),
				ct.getPrefix(),
				q);
	}
}
```

Σχήμα 44 ο κώδικας ανάκτησης δεδομένων της Ιδιότητας CourseTypes

```
public static List<CourseType> queryTypes() {
	return DB.Instance.q(CourseType.class, course, false);
}
```

Σχήμα 43 ο κώδικας της CoursesActions.queryTypes()

Η διαδικασία έχει ως εξής :

A. Ανακτούμε τα `CourseType` που σχετίζονται με το τρέχον μάθημα [`CoursesActions.queryTypes()`]
B. Για κάθε τύπο γίνονται τα εξής :
    I. Ανάκτηση των αντίστοιχων `CourseTypeRooms` [`DB.Instance.q(CourseTypeRoom.class, ct, true)`]
    II. Δημιουργία καινούργιας γραμμής στον πίνακα
        1. Το αντικείμενο `CourseTypes`
        2. Ο Τύπος Διδασκαλίας
        3. Οι ώρες που του αντιστοιχούν
        4. Το πρόθεμα
        5. Η λίστα με τ' αντίστοιχα `CourseTypesRoom`

3.10 Η διαδικασία των ελέγχων

Κεφάλαιο 4 - Συμπεράσματα

Τα αντικειμενοστρεφή συστήματα διαχείρισης βάσεων δεδομένων γενικότερα και ειδικότερα η εφαρμογή της db4o, παρουσιάζουν πολλαπλά πλεονεκτήματα συγκριτικά με τα σχεσιακά συστήματα, όπως έχει ήδη αναφερθεί.

Με τη χρήση της αντικειμενοστρεφούς βάσης δεδομένων db4o εξασφαλίζονται τόσο η ευκολία όσο και απλότητα σε ελάχιστες γραμμές κώδικα. Αυτή η απαράμιλλη ευκολία στη χρήση οδηγεί σε δραστική μείωση του χρόνου ανάπτυξης.

Έτσι, εξαλείφονται το σύνολο του έργου του σχεδιασμού, της υλοποίησης και της διατήρηση του σχήματος της βάσης δεδομένων, όπως και η ανάγκη για διαχείριση όλων όσων σχετίζονται με τις βάσεις δεδομένων, όπως strings, XML, ή άλλα μη-εγγενή αρχεία που χρειάζονται και κατά συνέπεια επιβραδύνουν τη διαδικασία παραγωγής.

Επιπλέον όμως, εξοικονομείται χρόνος και κάθε φορά που θα χρειαστεί να τροποποιηθεί το λογισμικό, π.χ. refactoring κώδικα, πρόσθεση νέων χαρακτηριστικών, ή επαναχρησιμοποίηση τμημάτων λογισμικού. Η αλλαγή του μοντέλου αντικειμένου, για παράδειγμα, δεν είναι μόνο εξαιρετικά διαφανής, αλλά αλάθητη επειδή η db4o έχει αναλάβει να κάνει όλο το έργο. Δεν απαιτούνται διορθωτές, καμία διαδικασία δόμησης, και δε χρειάζεται κάποια ανησυχία σχετικά με υφιστάμενες αναπτύξεις, διότι η db4o φροντίζει για τυχόν τροποποιήσεις του αντικειμένου στη βάση δεδομένων.

Στην ουσία, η db4o καθιστά τόσο εύκολη την ύπαρξη μόνιμων αντικειμένων, καθώς χρησιμοποιεί απλό serialization, αλλά και προσφέρει όλο το εύρος των λειτουργιών μίας απλής βάσης δεδομένων, όπως για παράδειγμα η επερώτηση (queries).

Επιπρόσθετα, η db4o έχει σχεδιαστεί έτσι ώστε να ενσωματώνεται σε εφαρμογές και να είναι εντελώς αόρατη για τον τελικό χρήστη. Και επειδή η db4o τρέχει στην ίδια διαδικασία με την εφαρμογή που τη χρησιμοποιεί, προσφέρεται πλήρης έλεγχος επί της διαχείρισης της μνήμης και μπορεί να εκτελέσει speed profiling και εντοπισμό σφαλμάτων σε όλο το σύστημα.

Από όλα τα παραπάνω είναι πολύ εύκολο να κατανοηθεί ο λόγος προτίμησης μίας αντικειμενοστρεφούς βάσης δεδομένων και συγκεκριμένα ο λόγος επιλογής της db4o.

Αφού εξηγήθηκαν οι λόγοι επιλογής της αντικειμενοστρεφούς βάσης db4o, θα γίνει κάποια παράθεση συμπερασμάτων σχετικά με τη δημιουργηθείσα εφαρμογή:

Η εν λόγω εφαρμογή επιτρέπει τη δημιουργία ωρολόγιου προγράμματος εξαμήνου, προσφέροντας στον χρήστη τη δυνατότητα να δημιουργεί και να τροποποιεί το πρόγραμμα αδιαφορώντας για τους όποιους περιορισμούς μπορεί να υπάρξουν (χρονικοί, χωροταξικοί, λογικοί, κ.ά.). Επομένως, αποτελεί μία χρήσιμη εφαρμογή καθώς η σύνταξη του προγράμματος εξαμήνου για μία σχολή αποτελεί μία χρονοβόρα διαδικασία και απαιτεί ιδιαίτερη προσοχή.

Η βασική ευκολία που προσφέρει αυτή η εφαρμογή είναι ότι πραγματοποιεί ελέγχους για το αν παραβιάζεται κάποιος εκ των περιορισμών (για παράδειγμα ένας καθηγητής να διδάσκει την ίδια ώρα σε διαφορετικές αίθουσες) και αναφέρει στον χρήστη πού εντοπίστηκε η παραβίαση. Γεγονός που είναι άκρως βοηθητικό για τον χρήστη διότι γνωρίζει πού βρίσκεται η παραβίαση και ποια είναι ώστε να τη διορθώσει.

Η παρούσα εφαρμογή αναφέρεται στο Τμήμα Μηχανικών Πληροφορικής του ΑΤΕΙΘ. Σαφώς, θα μπορούσε να γίνει επέκταση αυτής για ολόκληρο το εκπαιδευτικό ίδρυμα ΑΤΕΙΘ. Κάτι τέτοιο θα ήταν χρήσιμο, καθώς ένας καθηγητής δύναται να διδάσκει σε περισσότερα από ένα Τμήματα και πολλές φορές λόγω έλλειψης αιθουσών, πραγματοποιούνται μαθήματα και σε αίθουσες άλλων τμημάτων. Όπως όμως γίνεται εύκολα αντιληπτό, άμεσο αποτέλεσμα αυτής της επέκτασης, είναι η μεγέθυνση της βάσης δεδομένων καθώς και η αύξηση της πολυπλοκότητας της εφαρμογής.

Πέρα όμως από αυτές τις επεκτάσεις, η εφαρμογή θα μπορούσε να επεκταθεί και ως προς τη λειτουργικότητά της. Συγκεκριμένα, χρήσιμο θα ήταν ένα κουμπί «Αυτόματη Συμπλήρωση» που όταν το πατάει ο χρήστης να συμπληρώνεται το πρόγραμμα αυτόματα με όσα μαθήματα δεν έχουν εισαχθεί στο πρόγραμμα. Με αυτό τον τρόπο θα γινόταν σαφώς πιο γρήγορη η διαδικασία καταχώρησης.

Επιπλέον, μια άλλη επέκταση θα μπορούσε να θεωρηθεί η προβολή διαθέσιμων επιλογών στον χρήστη μετά την εφαρμογή των ελέγχων. Δηλαδή, όταν ο χρήστης επιλέγει την πραγματοποίηση ελέγχων και εμφανίζεται η λίστα με τους περιορισμούς που παραβιάστηκαν, να εμφανίζεται και μία λίστα με πιθανές λύσεις.

Βέβαια, αυτές είναι επεκτάσεις που δεν εμπίπτουν στις απαιτήσεις της παρούσας εργασίας, διότι βασικός σκοπός της είναι η υλοποίηση της εφαρμογής με τη βοήθεια της αντικειμενοστρεφούς βάσης δεδομένων db4o και η έμφαση στον τρόπο χρήσης της βάσης.

Η χρήση της αντικειμενοστρεφούς βάσης δεδομένων db4o μας έδωσε την δυνατότητα να ασχοληθούμε αποκλειστικά με την σχεδίαση της εφαρμογής και την συγγραφή του κώδικα. Η χρησιμοποίηση της προτείνεται ανεπιφύλακτα σε εφαρμογές που δεν έχουν εξειδικευμένες απαιτήσεις από την βάση δεδομένων.

Ο λόγος που δεν προτείνεται για κάθε είδους εφαρμογή έχουν να κάνουν ουσιαστικά με το ότι, παρότι έχει δημιουργηθεί από το 2000, η εξέλιξη της είναι μικρή. Οι σχεσιακές βάσεις δεδομένων εδώ και χρόνια έχουν εξελιχθεί σε κάτι παραπάνω από μια απλή αποθήκη δεδομένων. Προσφέρουν δυνατότητες και εργαλεία που είναι επιθυμητά σε πολλές περιπτώσεις χρήσης. Τόσο επιθυμητά που τελικά αντισταθμίζει την επιπλέον δουλειά που χρειάζονται. Η χρήση της db4o και γενικά των αντικειμενοστρεφών βάσεων δεν είναι αμελητέα. Έχει μέλλον ακόμα η τεχνολογία αυτή και με λίγα χρόνια ανάπτυξης θα μπορέσει να ανταγωνιστεί επάξια της σχεσιακές.

Το μεγάλο προσόν της db4o, όπως και γενικά των αντικειμενοστρεφών βάσεων δεδομένων, είναι η ταχύτητα. Η απόκρισή της σε ανάκτηση και αποθήκευση δεδομένων είναι πολλή μικρότερη από οποιαδήποτε άλλη βάση δεδομένων. Αν βασικό μας θέμα είναι η ταχύτητα και η κατανάλωση μνήμης στο τερματικό που θα δουλεύει η εφαρμογή, τότε η db4o είναι ουσιαστικά μονόδρομος. Η db4o μπορεί να χρησιμοποιηθεί σε Java και C# εφαρμογές και αυτό σημαίνει ότι μπορεί να χρησιμοποιηθεί σε εφαρμογές που προορίζονται για κινητές συσκευές (φορητοί υπολογιστές, android συσκευές). Η απόδοση σε τέτοιες συσκευές σε ζητήματα μνήμης και ταχύτητας είναι κρίσιμη.

Βιβλιογραφία

1. R. Elmasri και S.B. Navathe, 2007. ***Θεμελιώδεις Αρχές Συστημάτων Βάσεων Δεδομένων, (5η έκδοση αναθεωρημένη),*** τόμος Α και Β, Εκδόσεις Δίαυλος.

2. R. Ramakrishnan, J. Gehrke, 2012. ***Συστήματα Διαχείρισης Βάσεων Δεδομένων, 3η Έκδοση***, Εκδόσεις Α. ΤΖΙΟΛΑ.

3. Silberschatz, H. F. Korth, S. Sudarshan, 2010. *Database System Concepts,* William C Brown Pub.

4. H. Schildt, 2007. ***Swing A Beginner's Guide***, McGraw Hill.

5. J. Paterson, S. Edlich, H. Hörning, και R. Hörning, 2006. ***The Definitive Guide to db4o,*** Apress.

6. R.G.G.Cattell et. Al., 2000. ***The Object Data Standard, ODMG 3.0,*** Morgan Kaufmann Publishers.

7. Db4o Tutorial (Java), http://community.versant.com/Documentation/Reference/db4o-8.0/java/tutorial/
8. Db4o Reference (Java), http://community.versant.com/Documentation/Reference/db4o-8.0/java/reference/
9. Operational Database Management Systems, http://www.odbms.org/odmg/
10. Object-Oriented Databases Oriented Databases, db4o: Part 1, http://www.odbms.org/wp-content/uploads/2013/11/035.03-Grossniklaus-ODBMS-Lecture-db4o-Part1.2009.pdf
11. A Persistence Service for the OSGi framework, http://community.versant.com/Portals/0/resources/communitycontent/presentations/PersistenceForOSGi_CRosenberger.pdf
12. Db4o, http://en.wikipedia.org/wiki/Db4o
13. Συναλλαγές (βάσεις δεδομένων) http://el.wikipedia.org/wiki/Συναλλαγές_(βάσεις_δεδομένων)
14. db4o :: Java & .NET Object Database :: Object Source Persistence, Object Database :: Features and Benefits http://www.db4o.com/about/productinformation/features/