Τ.Ε.Ι. ΗΠΕΙΡΟΥ
Τμήμα τηλεπληροφορικής και διοίκησης.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΧΕΔΙΑΣΗ - ΥΛΟΠΟΙΗΣΗ ΟΛΟΚΛΗΡΩΜΕΝΗΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΕΦΑΡΜΟΓΗΣ ΠΑΡΟΧΗΣ ΥΠΗΡΕΣΙΩΝ ΤΗΛΕΚΠΑΙΔΕΥΣΗΣ

Σπουδαστές: ΚΟΥΚΟΥΡΟΥΖΗΣ ΝΙΚΟΛΑΟΣ
ΧΑΤΖΟΓΛΟΥ ΓΕΩΡΓΙΟΣ

Επιβλέπων καθηγητής: ΚΟΝΕΤΑΣ ΔΗΜΗΤΡΗΣ

Άρτα 2003

ΠΡΟΛΟΓΟΣ

Η αλματώδης ανάπτυξη των δικτύων και των τηλεπικοινωνιών την τελευταία δεκαετία έχουν ανοίξει καινούριους ορίζοντες και έχουν προσφέρει καινούριες δυνατότητες. Τα δίκτυα υπολογιστών έχουν εκμηδενίσει τις αποστάσεις και προσφέρουν καινούριους τρόπους επικοινωνίας. Είναι προφανές ότι οι δυνατότητες αυτές δεν θα άφηναν ασυγκίνητο το χώρο της εκπαίδευσης. Οι νέες τεχνολογίες μπορούν να αποτελέσουν ένα ισχυρό εργαλείο για την ενδυνάμωση της εκπαιδευτικής διαδικασίας και να δώσουν μία άλλη διάσταση στη μάθηση.

Η παρούσα πτυχιακή εργασία πραγματεύεται τη σχεδίαση και υλοποίηση ενός συστήματος παροχής υπηρεσιών Τηλεκπαίδευσης με την εκμετάλλευση των νέων αυτών τεχνολογιών που μας προσφέρει ο κόσμος του Διαδικτύου. Βασικός στόχος της εργασίας είναι η κατασκευή ενός δυναμικού εργαλείου Τηλεκπαίδευσης το οποίο να μπορεί να χρησιμοποιηθεί για κάθε ανάγκη απομακρυσμένης εκπαίδευσης.

ΠΕΡΙΕΧΟΜΕΝΑ

Κεφάλαιο 5

Λειτουργικός Σχεδιασμός Συστήματος

Κεφάλαιο 6

Υλοποίηση Συστήματος Τηλεκπαίδευσης (Virtual Class)

Κεφάλαιο 7

Client – Server Programming

Κεφάλαιο 1^{ο} : Η Εκπαίδευση από Απόσταση

1.1 Η Εκπαίδευση από Απόσταση ή Τηλεκπαίδευση

Η Εκπαίδευση από Απόσταση ή Τηλεκπαίδευση είναι μία μορφή ελεύθερης εκπαίδευσης στην οποία δεν απαιτείται ο εκπαιδευτής και οι εκπαιδευόμενοι να βρίσκονται στον ίδιο τόπο. Ο εκπαιδευτής επικοινωνεί με τους εκπαιδευόμενους με κάποιο μέσο αμφίδρομης επικοινωνίας, σύγχρονης ή ασύγχρονης. Η εκπαίδευση αυτή ονομάζεται και τηλε-επιμόρφωση ή τηλε-εκπαίδευση. Υπάρχουν πολλές μορφές εκπαίδευσης από απόσταση. Κάποιες από αυτές κάνουν προσομοίωση της διδασκαλίας που γίνεται μέσα σε μια τάξη με πλήρη επικοινωνία καθηγητών και μαθητών σε πραγματικό χρόνο, ενώ άλλες υποστηρίζουν την αυτορυθμιζόμενη μάθηση που κατευθύνεται από τον εκπαιδευόμενο. Η μορφή αυτορυθμιζόμενης μάθησης με ασύγχρονη επικοινωνία εφαρμόζεται στα περισσότερα συστήματα εκπαίδευσης από απόσταση. Ένας από τους στόχους της ανοιχτής και εξ αποστάσεως εκπαίδευσης είναι να παρέχει δυνατότητα πρόσβασης σε όλα τα επίπεδα εκπαίδευσης σε άτομα που δεν μπορούν με άλλους τρόπους να συμμετέχουν σε αυτά, λόγω της γεωγραφικής θέσης που κατοικούν ή λόγω ειδικών προσωπικών προβλημάτων. Άλλος στόχος είναι να μεταδοθούν μαθήματα σε απομακρυσμένες περιοχές στις οποίες δεν μπορούν να μεταβούν οι καθηγητές για να διδάξουν ή να. μεταδοθούν στα εκπαιδευτικά ιδρύματα μιας περιοχής μαθήματα στα οποία διδάσκουν διάσημοι καθηγητές από γνωστά πανεπιστήμια από όλο τον κόσμο. Η παρουσίαση μαθημάτων από απόσταση μπορεί να χρησιμοποιηθεί και για να βελτιώσει ένας καθηγητής τις τεχνικές διδασκαλίας του παρακολουθώντας άλλους καθηγητές να διδάσκουν το ίδιο μάθημα με αυτόν, ή για συνεργασία του καθηγητή με άλλους καθηγητές και για συνεργασία σχολείων μεταξύ τους.

Στο παρελθόν υπήρχε εκπαίδευση από απόσταση που γινόταν κυρίως δια αλληλογραφίας. Για τον ίδιο σκοπό οι εκπαιδευτές χρησιμοποιούσαν κασέτες ήχου και βιντεοκασέτες που αποστελλόταν ταχυδρομικά στους εκπαιδευόμενους. Επίσης γινόταν και χρήση καναλιών της τηλεόρασης όπου παρουσιαζόταν σεμινάρια και κύκλοι μαθημάτων με μορφή τηλεοπτικών εκπομπών. Όλα τα μέσα αυτά λέγονται μη

αλληλεπιδραστικά διότι δεν υπήρχε η δυνατότητα να απαντήσει άμεσα ο εκπαιδευόμενος.

Στη σημερινή εποχή έχουν αναπτυχθεί τα δίκτυα υπολογιστών που προσφέρουν πολλές δυνατότητες αλληλεπιδραστικής επικοινωνίας και διευκολύνουν την εκπαίδευση από απόσταση. Όλες οι πληροφορίες που βρίσκονται σε μορφή κειμένων, εικόνας και ήχου μετατρέπονται σε ψηφιακή μορφή. Μέσω του δικτύου υπολογιστών ο εκπαιδευτής μπορεί να αποστείλει τέτοιες πληροφορίες ψηφιακής μορφής στους εκπαιδευόμενους, οι οποίοι βρίσκονται σε μακρινές αποστάσεις. Το δίκτυο υπολογιστών είναι ένα μέσο επικοινωνίας, σύγχρονης ή ασύγχρονης. Αυτό το μέσο μπορεί να συνδυαστεί και με άλλα μέσα επικοινωνίας όπως είναι η αμφίδρομη τηλεόραση (interactive TV, ITV) ή η τηλεδιάσκεψη με φωνή (audio) και εικόνα (video) μέσω του Internet. Προγράμματα όπως το CU-SeeMe, NetMeeting, ClassPoint μπορούν να χρησιμοποιηθούν για μετάδοση video και audio σε πραγματικό χρόνο.

Με το δίκτυο υπολογιστών σε μία τηλεδιάσκεψη πολλών ατόμων μπορούν να γίνονται παρουσιάσεις κειμένων εικόνων, γραφικών και ήχου, να σχεδιάζονται παρουσιάσεις μαθημάτων με πολυμέσα (multimedia courses). Τα πολυμέσα παρουσιάζουν στον υπολογιστή κείμενα, προγράμματα software, εικόνες video και ήχου και με αυτά μπορεί να σχεδιαστεί εκπαιδευτικό λογισμικό (educational software). Στην εκπαίδευση εξ' αποστάσεως χρησιμοποιούνται αυτά τα μέσα επικοινωνίας σε συνδυασμό μεταξύ τους ώστε να υπάρξει όσο το δυνατόν καλύτερη καθοδήγηση των εκπαιδευόμενων. Ο όρος "Distributed online Education", σημαίνει τον συνδυασμό τεχνολογιών μετάδοσης πληροφοριών για διδασκαλία και μάθηση [11].

1.2 Σύγχρονη Εκπαίδευση

Στην σύγχρονη εκπαίδευση την ίδια χρονική στιγμή όλοι οι εκπαιδευόμενοι μαζί με τον εκπαιδευτή τους πρέπει να είναι συνδεδεμένοι στο δίκτυο και η επικοινωνία γίνεται σε πραγματικό χρόνο. Αυτή η μορφή εκπαίδευσης μπορεί να επιτευχθεί είτε με τηλεδιάσκεψη μέσω του δικτύου υπολογιστών, είτε με χρήση της αμφίδρομης τηλεόρασης ή με video-διάσκεψη μέσω του Internet[11]. Με το δίκτυο υπολογιστών μπορούν να μεταφέρονται εικόνες και ήχοι σε ψηφιακή μορφή, αρχεία εικόνας (video) και ήχου (audio). Υπάρχει η δυνατότητα μετάδοσης εικόνας (video) και ήχου (audio) σε πραγματικό χρόνο με προγράμματα όπως το real player. Μία

μορφή επικοινωνίας σε πραγματικό χρόνο είναι το πρόγραμμα IRC και τα παρόμοια προγράμματα talker's και chat's του Internet, όπως και τα MUDs και MOOs που επιτρέπουν την ταυτόχρονη επικοινωνία πολλών χρηστών του δικτύου με γραπτά μηνύματα. Επίσης τα προγράμματα talk, ICQ, write που επιτρέπουν την ταυτόχρονη επικοινωνία δύο χρηστών σε πραγματικό χρόνο με γραπτά μηνύματα. Στη σύγχρονη εκπαίδευση ανήκει και η video-διάσκεψη μέσω Internet (desktop videoconference) και η επικοινωνία CU-SeeMe , που επιτρέπουν επικοινωνία με σήμα video και ήχου. Η διδασκαλία μέσω Internet, σύμφωνα με έρευνες, για να έχει αποτελεσματικότητα απαιτεί συχνή αλληλεπιδραστική επικοινωνία σε πραγματικό χρόνο του καθηγητή με τους μαθητές και των μαθητών μεταξύ τους ώστε οι μαθητές να δέχονται συμβουλές και καθοδήγηση και να ενθαρρύνονται να συμμετέχουν σε ομαδικές εργασίες. Χωρίς αυτή την επικοινωνία, η διδασκαλία απομονώνει τον μαθητή και γίνεται απρόσωπη.

1.3 Ασύγχρονη Εκπαίδευση

Η ασύγχρονη εκπαίδευση δεν απαιτεί την ταυτόχρονη συμμετοχή όλων των μαθητών και των καθηγητών την ίδια χρονική στιγμή, αλλά γίνεται με την μορφή ανακοινώσεων. Οι εκπαιδευόμενοι επιλέγουν την χρονική στιγμή που θα διαβάσουν τις οδηγίες του εκπαιδευτή οι οποίες παραμένουν αποθηκευμένες σε κάποια περιοχή.[11]. Η ασύγχρονη εκπαίδευση είναι πιο ευέλικτη από την σύγχρονη καθοδήγηση. Μερικές μορφές παλαιότερης ασύγχρονης εκπαίδευσης είναι τα μαθήματα σε κασέτες ήχου ή Video, ή τα μαθήματα δια αλληλογραφίας. Νεότερες μέθοδοι είναι να παραδίδονται μαθήματα χρησιμοποιώντας τις υπηρεσίες του δικτύου υπολογιστών, όπως είναι οι παρακάτω για το δίκτυο Internet:

- Το ηλεκτρονικό ταχυδρομείο (e-mail)
- Οι ομάδες συζητήσεων μέσω ηλεκτρονικού ταχυδρομείου (mailing lists)
- Τα συστήματα με πίνακες ανακοινώσεων (Bulletin Board systems BBS)
- Οι ομάδες συζητήσεων (newsgroups)
- Ο Παγκόσμιος Ιστός (WWW)

Σε επόμενο κεφάλαιο θα αναφερθούμε πιο αναλυτικά στις υπηρεσίες δικτύου υπολογιστών στην εκπαίδευση. Οι καθηγητές και αυτοί που σχεδιάζουν τους κύκλους μαθημάτων πρέπει να γνωρίζουν τις εφαρμογές εκπαίδευσης από απόσταση στο Internet για να διαλέξουν τις κατάλληλες μεθόδους και να σχεδιάσουν αποτελεσματικές παρουσιάσεις των μαθημάτων.

1.4 Σε ποίους Απευθύνεται η Εκπαίδευση από Απόσταση

Η εκπαίδευση από απόσταση απευθύνεται σε όλες τις βαθμίδες εκπαίδευσης και σε εργαζόμενους ενήλικους. Μπορεί να διαχωριστεί σε τρεις βασικές κατηγορίες:

- Εκπαίδευση και διαρκής κατάρτιση εργαζόμενων ενηλίκων
- Ανώτερη εκπαίδευση σε πανεπιστήμια και σε κολέγια
- Κατώτερη και μέση εκπαίδευση σε μαθητές δημοτικών σχολείων και γυμνασίων λυκείων.

Τα τελευταία χρόνια γίνονται προσπάθειες σε κάθε χώρα, ώστε τα σχολεία όλων των βαθμίδων εκπαίδευσης και τα πανεπιστήμια να συνδεθούν στο παγκόσμιο δίκτυο υπολογιστών Internet. Έτσι η εκπαίδευση από απόσταση μέσω του Διαδικτύου μπορεί να είναι προσιτή σε μαθητές κάθε ηλικίας από όλο τον κόσμο. Οι ενήλικοι που παρακολουθούν μαθήματα με εκπαίδευση από απόσταση συνήθως είναι εργαζόμενοι ή έχουν οικογενειακές υποχρεώσεις και δεν μπορούν να παρακολουθήσουν κανονικά παραδόσεις μαθημάτων σε διδασκαλία πρόσωπο με πρόσωπο. Στην εκπαίδευση από απόσταση με ανεξάρτητη μάθηση οι σειρές μαθημάτων παραδίδονται στο χώρο και στο χρόνο που επιλέγει ο εκπαιδευόμενος, συνήθως στο σπίτι ή στο χώρο εργασίας του. Έτσι οι εργαζόμενοι προτιμούν να παρακολουθούν μαθήματα από απόσταση. Ακόμη όταν οι εργοδότες θέλουν να βελτιώσουν την μόρφωση των υπαλλήλων τους με επιμορφωτικά σεμινάρια, χωρίς όμως οι υπάλληλοι να φύγουν από τον τόπο εργασίας τους, επιλέγουν την εκπαίδευση από απόσταση.

Όταν η εκπαίδευση απευθύνεται σε μαθητές σχολείων χρησιμοποιούνται μέθοδοι προσομοίωσης της πραγματικής διδασκαλίας. Οι μαθητές επικοινωνούν με άλλα σχολεία της περιοχής ή με μαθητές από όλο τον κόσμο και συμμετέχουν σε ομαδικές εργασίες. Τα σχολεία απομακρυσμένων περιοχών παρακολουθούν μαθήματα από καθηγητές που βρίσκονται σε κεντρικές πόλεις και δεν μπορούν να επισκεφθούν τις περιοχές αυτές. Αυξάνονται οι ευκαιρίες επικοινωνίας και αυξάνεται η συμμετοχή των ατόμων που εκπαιδεύονται.

Στις Η.Π.Α, τα συστήματα εκπαίδευσης από απόσταση βελτιώνονται συνεχώς και εφαρμόζονται σε όλους τους τομείς της εκπαίδευσης. Συγκεκριμένα, κολέγια, ανοιχτά πανεπιστήμια, εικονικά-δυνητικά πανεπιστήμια, δημοτικά σχολεία και γυμνάσια (K-12), ιδρύματα δια βίου εκπαίδευσης ενηλίκων και διαρκούς κατάρτισης, παραδίδουν μαθήματα ή σεμινάρια μέσω του Internet και μέσω ψηφιακής τηλεόρασης σε μαθητές από όλο τον κόσμο. Σύντομα, το ίδιο αναμένεται να γίνει

στην Ελλάδα και στις άλλες χώρες της Ευρωπαϊκής ένωσης με την σύνδεση δημοτικών σχολείων, γυμνασίων, λυκείων και πανεπιστημίων κάθε χώρας στο Internet.

Η εκπαίδευση από απόσταση επίσης απευθύνεται και σε ειδικές κατηγορίες ατόμων με κινητικά προβλήματα, που δεν μπορούν να βγουν από το σπίτι και να παρακολουθήσουν κανονικά μαθήματα στο σχολείο. Για τα άτομα αυτά δημιουργούνται ειδικές υπηρεσίες χειρισμού του υπολογιστή, όπως ανίχνευση της κίνησης των ματιών του χρήστη με ενσωματωμένη κάμερα στον υπολογιστή και αυτόματη ενεργοποίηση των εντολών χωρίς να χρειάζεται να χρησιμοποιήσει ο χρήστης ποντίκι ή πληκτρολόγιο. Τα άτομα με ειδικές ανάγκες μπορούν να έχουν ισότιμη πρόσβαση στην εκπαίδευση και να παρακολουθούν το μάθημα μιας τάξης από απόσταση. Όταν η επικοινωνία γίνεται με γραπτά κείμενα, αυτό ενισχύει την ανωνυμία των εκπαιδευόμενων και έτσι μπορούν να αποκρύψουν από τους άλλους ότι είναι τυφλοί ή έχουν κινητικά προβλήματα και να συμμετέχουν ως ίσοι στην επικοινωνία. Ένα τεχνολογικό μέσο που μπορεί να χρησιμοποιήσει ένας τυφλός είναι ένα σύστημα που συνθέτει φωνή και του ανακοινώνει τα μηνύματα που εμφανίζονται στην οθόνη, ή μπορεί να χρησιμοποιήσει ζωντανή επικοινωνία με ήχο μέσω του Internet. Η ισότητα δίνει μεγαλύτερη αυτοπεποίθηση στο άτομο με ειδικές ανάγκες. Του δίνει δυνατότητα να επικοινωνήσει με άτομα που ίσως θα δίσταζαν να επικοινωνήσουν μαζί του πρόσωπο με πρόσωπο, διότι συχνά τα άτομα με ειδικές ανάγκες απομονώνονται από τους άλλους και αποτελούν ξεχωριστή μειονότητα. Η εκπαίδευση από απόσταση δίνει στους μαθητές με ειδικά προβλήματα ένα περιβάλλον στο οποίο έχουν αποτελεσματική επικοινωνία με ειδικούς καθηγητές που τους βοηθούν να υπερνικήσουν τις φυσικές δυσκολίες και να αποκτήσουν πλήρη εκπαίδευση. Δημιουργούνται κοινότητες ατόμων με ειδικές ανάγκες και μπορούν να εκπαιδευτούν μαζί ομάδες ατόμων που αντιμετωπίζουν κοινά προβλήματα. Το άτομο με ειδικές ανάγκες έχει πρόσβαση στις νέες τεχνολογίες και έχει στη διάθεση του ένα πλήθος προγραμμάτων και υπηρεσιών. Ψυχολόγοι και σύμβουλοι που κατοικούν σε μακρινές περιοχές, επικοινωνούν μαζί του και του παρέχουν οδηγίες από απόσταση.

1.5 Οι Διαφορές Εικονικών - Δυνητικών Πανεπιστημίων και Ανοιχτών Πανεπιστημίων

Σύμφωνα με τον Peraya μπορούμε να διακρίνουμε δύο διαφορετικές κατηγορίες ενηλίκων που αντιστοιχούν σε δύο διαφορετικά μοντέλα εκπαίδευσης από απόσταση. Τα δύο μοντέλα αυτά είναι το Εικονικό - Δυνητικό Πανεπιστήμιο (Virtual University) και το Ανοιχτό Πανεπιστήμιο (Open University).

Υπάρχουν ενήλικοι που αναζητούν πλήρη μόρφωση. Ενδιαφέρονται να παρακολουθήσουν ένα ολοκληρωμένο πρόγραμμα σπουδών και να αποκτήσουν ένα νέο πτυχίο. Έτσι η εκπαίδευση από απόσταση εμφανίζεται ως μια δεύτερη ευκαιρία εκπαίδευσης. Πολλοί είναι εργαζόμενοι και επαγγελματίες και η εκπαίδευση από απόσταση ίσως είναι ο μόνος τρόπος για να αποκτήσουν το πτυχίο που θέλουν.

Από την άλλη πλευρά υπάρχουν ενήλικοι που επιθυμούν απλά να αυξήσουν τις γνώσεις τους και να έχουν κάποια πρακτική εξάσκηση πάνω σε τομείς που αφορούν το επάγγελμα τους. Ενδιαφέρονται για συγκεκριμένο θέμα ή τεχνικές γνώσεις και θέλουν να παρακολουθήσουν ειδικά σεμινάρια. Αυτοί δεν είναι απαραίτητο να παρακολουθήσουν ένα ολοκληρωμένο πρόγραμμα σπουδών που θα τους παρέχει πτυχίο. Ο κύριος στόχος τους είναι να έχουν κάποια επιπλέον ενημέρωση που θα αυξήσει τα επαγγελματικά τους προσόντα [10].

Δημιουργούνται δύο μορφές εκπαίδευσης από απόσταση για να καλυφθούν οι ανάγκες αυτών των δύο κατηγοριών. Σε αυτές τις κατηγορίες εκπαίδευσης από απόσταση υπάρχει διαφορά ως προς τον ρόλο και τα καθήκοντα του καθηγητή. Η εκπαίδευση από απόσταση που παρουσιάζει ομοιότητες με εκπαίδευση συνηθισμένου πανεπιστημίου, γίνεται σε εικονικό-δυνητικό πανεπιστήμιο όπου απαιτούνται πανεπιστημιακοί καθηγητές και διεξαγωγή έρευνας παράλληλα με την διδασκαλία. Αντίθετα σε ένα ανοιχτό πανεπιστήμιο ο κύριος σκοπός είναι η διδασκαλία και η μετάδοση γνώσεων και το ερευνητικό έργο δεν είναι υποχρεωτικό για τους καθηγητές αλλά γίνεται προαιρετικά Αυτές οι γενικές διαφορές έχουν ως αποτέλεσμα να διαφέρουν στους δύο τρόπους εκπαίδευσης η οργάνωση της διδασκαλίας και η δημιουργία των διδακτικών κειμένων[10]. Στο Εικονικό - Δυνητικό Πανεπιστήμιο (Virtual University) για να παρακολουθήσει κανείς τα μαθήματα πρέπει να έχει απολυτήριο λυκείου ενώ στο ανοιχτό πανεπιστήμιο (Open University) μπορεί να εγγραφεί οποιοσδήποτε ενήλικος ανεξάρτητα από το επίπεδο μόρφωσης του.

Κεφάλαιο 2^{ο} : Αντικειμενοστραφής προγραμματισμός

2.1. Αντικειμενοστραφής Γλώσσα Προγραμματισμού Java

Η Java είναι σχεδιασμένη για το Internet. προγράμματα της Java μπορούν να τρέχουν σε προγράμματα πλοήγησης εφοδιασμένα με την κατάλληλη JVM (Java Virtual Machine). Αφού το πρόγραμμα κατέβει από το Internet εκτελείται στη μηχανή του πελάτη και όχι στο Server. Αυτά τα προγράμματα της Java είναι γνωστά σαν Java Applets.

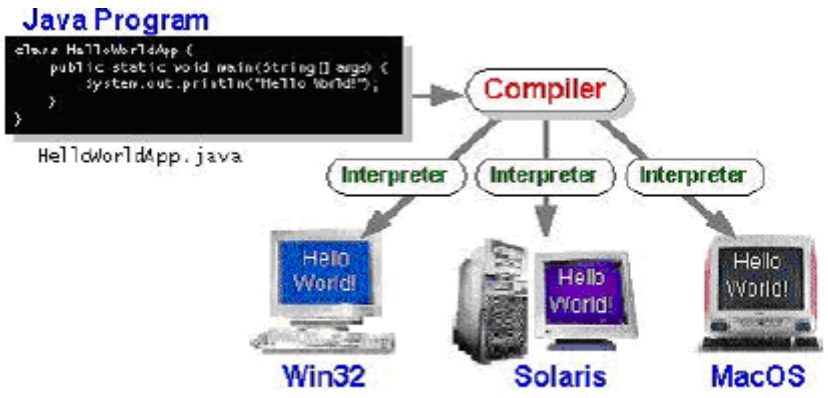
Η Java είναι σχεδιασμένη με στόχο την εξασφάλιση ασφάλειας. Ένα Java Applet εκτελείται με απόλυτη ασφάλεια στη μηχανή του χρήστη και δεν μπορεί να κάνει τίποτα που να μην επιτρέπεται από την πολιτική ασφάλειας της JVM της μηχανής του πελάτη. Η Java προσφέρει εκτέλεση σε όλους τους υπολογιστές. Με το επίπεδο αφαίρεσης υλικού της JVM ένα πρόγραμμα Java εκτελείται σε όλους τους υπολογιστές ανεξαρτήτως Λειτουργικού Συστήματος. Αρκεί αυτές να είναι εφοδιασμένες με την κατάλληλη JVM. Η Java είναι μια πλούσια και πολύ ισχυρή αντικειμενοστραφής γλώσσα προγραμματισμού για το Internet (και όχι μόνο) και όχι μία απλή script γλώσσα για την σύνθεση συστατικών (π.χ. VBScript με ActiveX controls). Αυτό μας επιτρέπει να γράφουμε προγράμματα τα οποία είναι εύκολο να συντηρηθούν και να εξελιχτούν στο χρόνο και όχι απλά σενάρια (scripts) τα οποία παίζουν το ρόλο της κόλλας για components που έχουν γίνει με άλλες γλώσσες αλλά γενικά δεν μπορούμε να τα αλλάξουμε ή να ελέγξουμε την εξέλιξή τους.

Η Java προσφέρει την ευελιξία της δυναμικής σύνδεσης. Ένα πρόγραμμα Java εκτελείται από μία εικονική μηχανή και οι τάξεις (τα αρχεία .class που περιέχουν τα bytecodes) φορτώνονται από την JVM όταν εκτελείται το πρόγραμμα. Αυτό σημαίνει ότι μπορεί κανείς να αντικαταστήσει μία παλιά τάξη με μία νέα ακόμα και την στιγμή που εκτελείται το πρόγραμμα. Αυτό είναι ένα βασικό χαρακτηριστικό των components. Έτσι οι τάξεις Java πάνε πέρα από τον αντικειμενοστραφή προγραμματισμό, στον προγραμματισμό βασισμένο σε συστατικά (component-based programming). Είναι επίσης σημαντικό ότι η Java είναι σχεδιασμένη γι’ αυτό και δεν απαιτείται η προσθήκη κάποιου πολύπλοκου μοντέλου (όπως το COM) της

Microsoft, πάνω από μία γλώσσα προγραμματισμού που δεν είναι σχεδιασμένη με αυτό τον τρόπο (π.χ. Visual C++).

2.2 Java και Ανεξαρτησία από μηχανή

Ο μεταγλωττιστής της Java δεν παράγει εκτελέσιμο κώδικα αλλά παράγει μία μορφή αρχείων γνωστή ως Java Bytecodes που δεν είναι εκτελέσιμη απευθείας από μία μηχανή αλλά εκτελείται από την εικονική μηχανή Java (Java Virtual Machine – JVM). Αυτό δίνει στην Java το πλεονέκτημα της ανεξαρτησίας από την συγκεκριμένη μηχανή στην οποία εκτελείται. Αρκεί κανείς να έχει εγκαταστήσει την κατάλληλη έκδοση της Java και το πρόγραμμα θα εκτελεστεί από την εικονική μηχανή. Αυτή η επινόηση έμεινε γνωστή σαν «Γράψε μια φορά για να τρέχει παντού» (Write Once – Run Everywhere). Η εικόνα 2.1 μας δείχνει την ιδέα της εικονικής μηχανής Java. Όπως βλέπουμε στην εικόνα το ίδιο πρόγραμμα αφού μεταφραστεί σε bytecodes στη συνέχεια μπορεί να τρέξει σε διαφορετικές μηχανές (π.χ Windows, Solaris ή MacOS). Αρκεί το σύστημα να είναι εφοδιασμένο με την κατάλληλη JVM που θα το εκτελέσει.



Εικόνα 2.1 Εικονική Μηχανή Java

Κεφάλαιο 3^{ο} : Επεξήγηση εννοιών

3.1 Αντικειμενοστραφής Προγραμματισμός

Ο Αντικειμενοστραφής Προγραμματισμός (Object Oriented Programming, OOP) είναι ένας τρόπος συγγραφής προγραμμάτων, ο οποίος βασίζεται στην θεωρία πως οποιαδήποτε οντότητα σε ένα πρόγραμμα μπορεί να μοντελοποιηθεί με έναν τρόπο πολύ οικείο στον άνθρωπο, αυτόν *του* αντικειμένου.

3.2 Αντικείμενο

Αντικείμενο ονομάζεται κάθε συλλογή δεδομένων και διαδικασιών, η οποία έχει δημιουργηθεί με σκοπό να παρουσιάζει κάποια συγκεκριμένη συμπεριφορά. Ένα αντικείμενο καθορίζεται από την κατάσταση του και την συμπεριφορά του. Η κατάσταση στην οποία βρίσκεται περιγράφεται από τις μεταβλητές που περιέχει, ενώ η συμπεριφορά του από τις μεθόδους**.** Το αντικείμενο χρησιμοποιείται από τις αντικειμενοστραφείς γλώσσες προγραμματισμού με τον όρο της Τάξης (class).

Γενικά μία τάξη (class) είναι ένα καλούπι από το οποίο δημιουργούνται αντικείμενα.

Μία τάξη είναι μια μονάδα:

- Αφαίρεσης: διότι παριστάνει μία οντότητα του χώρου λύσης του προβλήματος
- Συγκέντρωσης: διότι συγκεντρώνει δεδομένα και μεθόδους κάτω από τον ίδιο τύπο.
- Απόκρυψης: διότι επιτρέπει τον απόλυτο έλεγχο στο ποιες άλλες τάξεις μπορούν να προσπελάσουν τα δεδομένα της
- Επέκτασης: διότι μπορεί να αποτελέσει την βάση ή την επέκταση ή και τα δύο κάποιας ή κάποιων άλλων τάξεων.
- Εξέλιξης: διότι δεδομένων κάποιων προϋποθέσεων μπορεί να διευκολύνει στην εξέλιξη του λογισμικού με συστηματικό τρόπο

3.3 Κληρονομικότητα

Ο όρος κληρονομικότητα προέρχεται από το γεγονός ότι μια τάξη που επεκτείνει μια υπάρχουσα τάξη κληρονομεί απ' αυτή όλες τις μεθόδους και τα

δεδομένα της, αλλά και τον τύπο της. δηλαδή, η νέα τάξη που δημιουργείται σαν επέκταση της παλιάς θα μπορεί να χρησιμοποιηθεί οπουδήποτε χρησιμοποιούταν η παλιά αφού το σύστημα τύπων (type system) δεν θα διαμαρτυρηθεί.

3.4 Διασύνδεση (Interface)

Διασύνδεση είναι μία συλλογή μεθόδων που δηλώνει ότι μία κλάση έχει μία συμπεριφορά επιπρόσθετα σε αυτή που κληρονομεί από τις υπερκλάσεις τις[1].

3.5 Κλήση Απομακρυσμένης Μεθόδου (Remote Method Invocation)

Η κλήση απομακρυσμένης μεθόδου (remote method invocation) δημιουργεί εφαρμογές της Java, που μπορούν να μιλήσουν με άλλες εφαρμογές επάνω σε ένα δίκτυο. Για να είμαστε πιο συγκεκριμένοι, το RMI επιτρέπει σε μια εφαρμογή να καλεί μεθόδους και να προσπελαύνει μεταβλητές μέσα σε μια άλλη εφαρμογή, που μπορεί να εκτελείται σε διαφορετικά περιβάλλοντα της Java η σε τελείως διαφορετικά λειτουργικά συστήματα, και να περνά από συνδέσεις δικτύου. Το RMI είναι ένας πιο έξυπνος μηχανισμός για επικοινωνία ανάμεσα σε κατανεμημένα αντικείμενα της Java από ότι είναι μια απλή σύνδεση Socket, επειδή οι μηχανισμοί και τα πρωτοκολλά με τα οποία επικοινωνείτε ανάμεσα σε αντικείμενα ορίζονται και προτυποποιούνται. Μπορείτε να συνομιλήσετε με ένα άλλο πρόγραμμα της Java χρησιμοποιώντας το RMI, χωρίς να χρειάζεται να ξέρετε εκ των προτέρων σε ποιο πρωτόκολλο θα επικοινωνήσετε ή με τι τρόπο. Ενώ η αρχή του RMI μπορεί να σας φέρνει στο μυαλό εικόνες αντικειμένων που επικοινωνούν μεταξύ τους, το RMI χρησιμοποιείται συνήθως σε πιο παραδοσιακές καταστάσεις πελάτη / διακομιστή. Μια εφαρμογή διακομιστή δέχεται συνδέσεις και αιτήσεις από ένα αριθμό πελατών. Το RMI είναι ένας απλός μηχανισμός με τον οποίο επικοινωνούν ο πελάτης και ο διακομιστής.

3.6 Κατανεμημένο Σύστημα Υπολογιστών (Distributed System)

Κατανεμημένο Σύστημα Υπολογιστών ονομάζεται κάθε σύστημα που περιλαμβάνει δύο ή περισσότερους ανεξάρτητους υπολογιστές που μπορούν να ανταλλάσσουν πληροφορίες και να συνεργάζονται μεταξύ τους.

Κεφάλαιο 4^{ο} : Περιγραφή Συστήματος Τηλεκπαίδευσης (Virtual Class)

4.1 Εισαγωγή στο Virtual Class

Το Virtual Class είναι ένα ολοκληρωμένο Σύστημα Παροχής Υπηρεσιών Τηλεκπαίδευσης. Είναι ένα Σύστημα που εκμεταλλεύεται τις ιδιότητες και τα πλεονεκτήματα των δύο βασικών Μοντέλων Τηλεκπαίδευσης, του Ανοιχτού Πανεπιστημίου (Open University) και του Εικονικού - Δυνητικού Πανεπιστημίου (Virtual University). Αυτό σημαίνει πως μπορεί κάποιος να χρησιμοποιήσει το Virtual Class ως ένα ολοκληρωμένο εργαλείο παροχής γνώσεων με τη μορφή πιστοποιημένων κύκλων τηλεμαθημάτων ή ως ένα εργαλείο μετάδοσης γνώσεων με μορφή σεμιναρίων. Ως μια πρωτεύουσα ή δευτερεύουσα αντίστοιχα, πηγή εκμετάλλευσης γνώσεων.

Στην ουσία το Virtual Class δεν αποτελεί υλοποίηση ενός από τα δύο Μοντέλα Τηλεκπαίδευσης αλλά, ένα εργαλείο Τηλεκπαίδευσης το οποίο χρησιμοποιεί βασικά χαρακτηριστικά των δυο. Είναι ένα Εικονικό Σύστημα Τηλεκπαίδευσης βασισμένο σ' ένα Συμβατικό Σύστημα Εκπαίδευσης πιο δυναμικό και ευέλικτο. Εκμεταλλεύοντας όρους, δομές και λειτουργίες ενός συμβατικού Πανεπιστημίου χωρίς να αποτελούνε τη βάση λειτουργίας του, επιτυγχάνει τις δύο κατευθύνσεις, των βασικών Μοντέλων Τηλεκπαίδευσης.

4.2 Virtual Class Εφαρμογή ή Εργαλείο Τηλεκπαίδευσης;

Το Virtual Class είναι και Εφαρμογή και Εργαλείο Τηλεκπαίδευσης. Αποτελείται από ένα σύνολο προγραμματιστικών βιβλιοθηκών σε γλώσσα Java μέσα στις οποίες υλοποιούνται Εργαλεία Τηλεκπαίδευσης και ένα ολοκληρωμένο Σύστημα Παροχής Υπηρεσιών Τηλεκπαίδευσης.

- **Δύο βιβλιοθήκες, Πηγές (Sources)**

  Learning Tools. Είναι η βιβλιοθήκη αυτή που περιέχει όλο το σύνολο των εργαλείων Τηλεκπαίδευσης. Τα εργαλεία αυτά είναι αντικείμενα βασισμένα σε εκπαιδευτικούς όρους και λειτουργίες μέσω των οποίων μπορεί να χρησιμοποιηθεί κατάλληλα ο Ηλεκτρονικός Υπολογιστής για Εκπαιδευτική χρήση. Το Learning Tools αποτελεί το ουσιαστικό προγραμματιστικό υλικό

μέσω του οποίου μπορεί να κατασκευαστεί ένα οποιοδήποτε Σύστημα Τηλεκπαίδευσης σε πλατφόρμα της επιλογής μας (Site, Application).

Learning Graphics. Είναι η βιβλιοθήκη αυτή που περιέχει ένα σύνολο γραφικών αντικειμένων και interfaces τα οποία ασχολούνται με το γραφικό περιβάλλον το οποίο παρεμβάλλεται ανάμεσα στο χρήστη και σε ένα Σύστημα Τηλεκπαίδευσης και με τον τρόπο με τον οποίο μπορεί να γίνει ευκολότερη η εκπαίδευση μέσω ενός Υπολογιστικού Συστήματος. Τα Components αυτά στην ουσία είναι Εργαλεία Τηλεκπαίδευσης (Learning Tools) κληροδοτημένα με ιδιότητες γραφικών αντικειμένων.

- **Μία βιβλιοθήκη που υλοποιεί ένα Σύστημα Τηλεκπαίδευσης (Applications)**

  Learning Class. Είναι η βιβλιοθήκη η οποία υλοποιεί το Σύστημα Τηλεκπαίδευσης Virtual Class βασισμένο στα προγραμματιστικά εργαλεία των δύο βασικών βιβλιοθηκών του (Learning Tools, Learning Graphics).

Η συνύπαρξη των δύο βιβλιοθηκών Learning Tools, Learning Graphics εξηγεί το γεγονός γιατί το Virtual Class ικανοποιεί και τα δύο Μοντέλα Τηλεκπαίδευσης μιας και το ίδιο το Virtual Class είναι μια βιβλιοθήκη διαθέσιμη για σχεδιασμό και υλοποίηση οποιοδήποτε Συστήματος Τηλεκπαίδευσης χρειαζόμαστε. Το Σύστημα το οποίο μας παρέχει το Learning Class είναι χαρακτηριστικό παράδειγμα αυτής της ιδιότητας.

4.3 Εκπαίδευση μέσα από το Σύστημα Learning Class

Στη συνέχεια εξετάσουμε τον τρόπο λειτουργίας του Συστήματος Τηλεκπαίδευσης Learning Class και τον τρόπο με τον οποίο μπορεί να επιτευχθεί Τηλεκπαίδευση μέσα από αυτό.

Το Learning Class είναι ένα Σύστημα Τηλεκπαίδευσης το οποίο υλοποιεί ένα συμβατό Πανεπιστήμιο στο Internet. Ο τρόπος διεξαγωγής της Εκπαίδευσης γίνεται με μορφή σημειώσεων και εργασιών. Οι σημειώσεις αποτελούν την ύλη και τον τρόπο παράδοσης ενός Τηλεμαθήματος. Ο Μαθητής επιλέγει την χρονική στιγμή που θα διαβάσει τις σημειώσεις ενός Μαθήματος οι οποίες παραμένουν αποθηκευμένες και διαθέσιμες για όλο τον κύκλο Μαθήματος. Ένας κύκλος μαθήματος αποτελείται από την παράδοση σημειώσεων, συμπληρωματικών εργασιών και τεστ μέσα σε μία συγκεκριμένη χρονική περίοδο. Το χρονικό διάστημα ενός κύκλου μαθήματος μπορεί

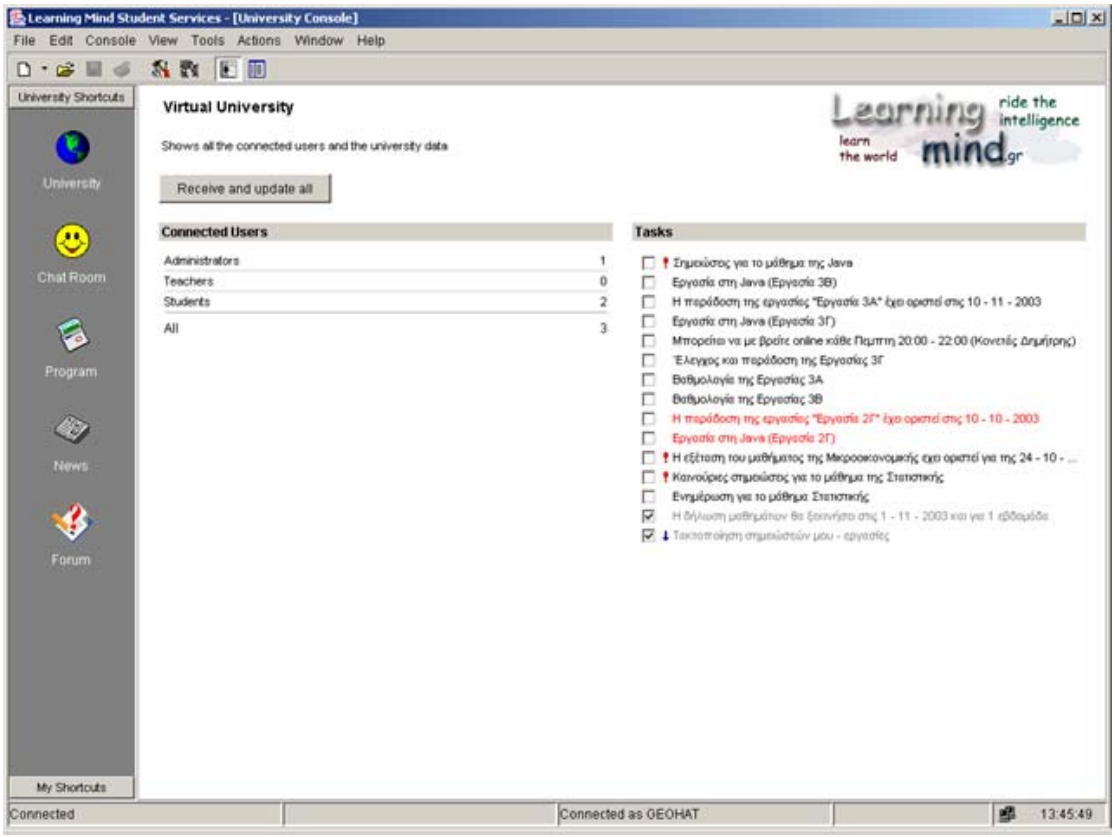
να μεταβάλλεται ανάλογα με το είδος της Εκπαίδευσης, ενώ για τις απαιτήσεις ενός Εικονικού Πανεπιστημίου χρησιμοποιούμε χρονικές περιόδους εξαμήνων.

Το σύστημα δεν ασχολείται με τον τρόπο αξιολόγησης των μαθητευομένων αλλά με την κατάλληλη χρήση των εργαλείων του κάτι τέτοιο θα μπορούσε να επιτευχθεί. Αυτό εξαρτάται κυρίως από την πολιτική του Εικονικού Πανεπιστημίου και το είδος της Εκπαίδευσης που θέλουμε να προσφέρει. Αναφορικά μπορούμε να πούμε πως σε κάθε περίπτωση η αξιολόγηση των μαθητευομένων είναι στη δικαιοδοσία των καθηγητών ο οποίος είναι υποχρεωμένος ή όχι να κρατάει καταστάσεις βαθμολογιών και να ενημερώνει τους μαθητές. Συμπερασματικά με τα όσα έχουμε προαναφέρει μπορούμε να πούμε πως το θέμα της Πιστοποίησης των γνώσεων μέσα από ένα τέτοιο Σύστημα δεν αφορά το ίδιο το Σύστημα αλλά εξαρτάται από τον τρόπο που αυτό θα χρησιμοποιηθεί.

Το Σύστημα χρησιμοποιεί κατά βάση μέσα και τρόπους ασύγχρονης εκπαίδευσης γι αυτό και δεν υπάρχουν περιορισμοί στα μαθήματα. Δηλαδή σε κάθε μάθημα μπορεί να εγγραφεί θεωρητικά οποιοσδήποτε αριθμός μαθητευομένων αφού περίπτωση ταυτόχρονης συνδιάσκεψης χρηστών γίνεται μόνο μέσω του Chat το οποίο χρησιμοποιεί ελάχιστους πόρους δικτύου.

4.4 Παρουσίαση Learning Class

Το Learning Class είναι ένα Σύστημα παροχής Υπηρεσιών Τηλεκπαίδευσης το οποίο είναι κατασκευασμένο σε παραθυρικό περιβάλλον εφαρμογής και υλοποιεί εργαλεία Σύγχρονης και Ασύγχρονης Εκπαίδευσης (Εικόνα 4.1).



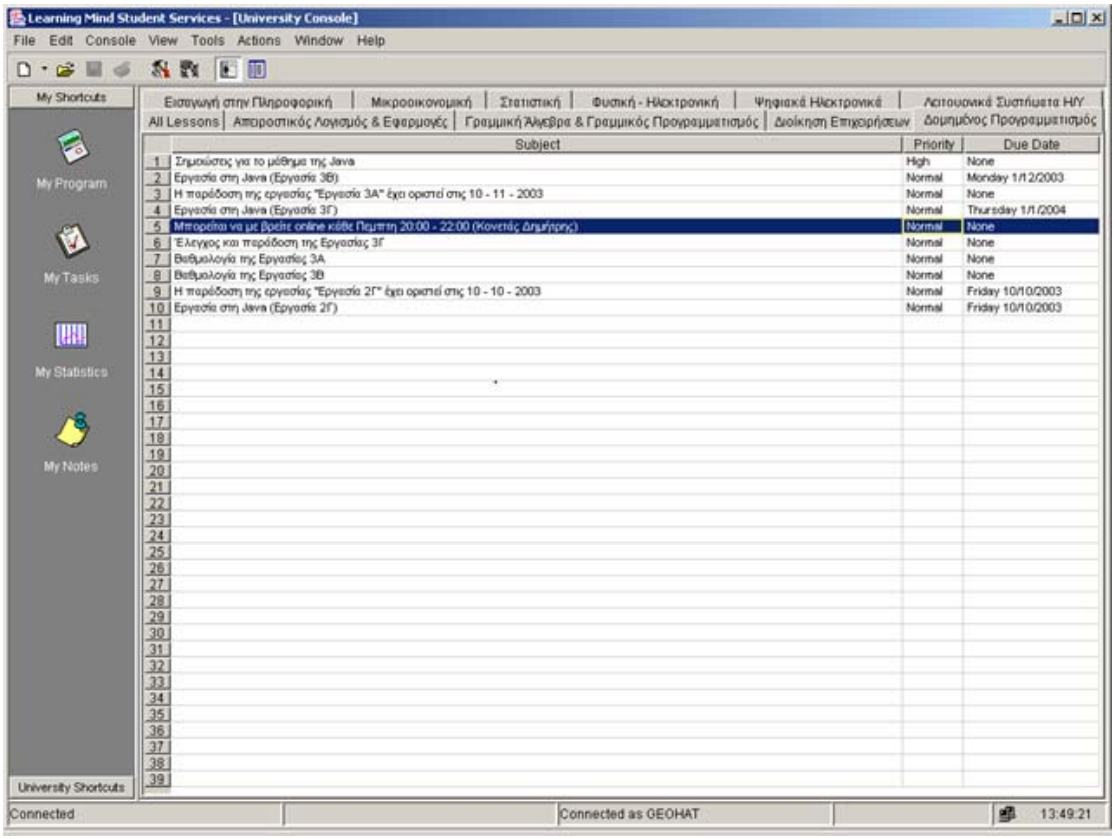
Εικόνα 4.1 Κεντρική κονσόλα Εφαρμογής Learning Class

Εργαλεία Ασύγχρονη Εκπαίδευση :

- Tasks (Εργασίες – Ανακοινώσεις). Τα Tasks είναι ένα εργαλείο το οποίο κατά κύριο λόγο το χρησιμοποιεί ο Καθηγητής για να δημιουργεί εργασίες στο Μάθημά του και να στέλνει ανακοινώσεις στους Μαθητές. Ο Μαθητής αντίστοιχα μέσω των Tasks ενημερώνεται για τις υποχρεώσεις του σε κάθε Μάθημα και για τις εργασίες τις οποίες πρέπει να ασχοληθεί. Ο Καθηγητής δημιουργεί Tasks για όλη την Τάξη του Μαθήματος ενώ ο Μαθητής χρησιμοποιεί τα Tasks για ιδιωτική χρήση.

  Στην εικόνα 4.2 βλέπουμε την κονσόλα διαχείρισης των Tasks. Στο παράθυρο φαίνονται όλα τα μαθήματα της τρέχουσας εκπαιδευτικής περιόδου για

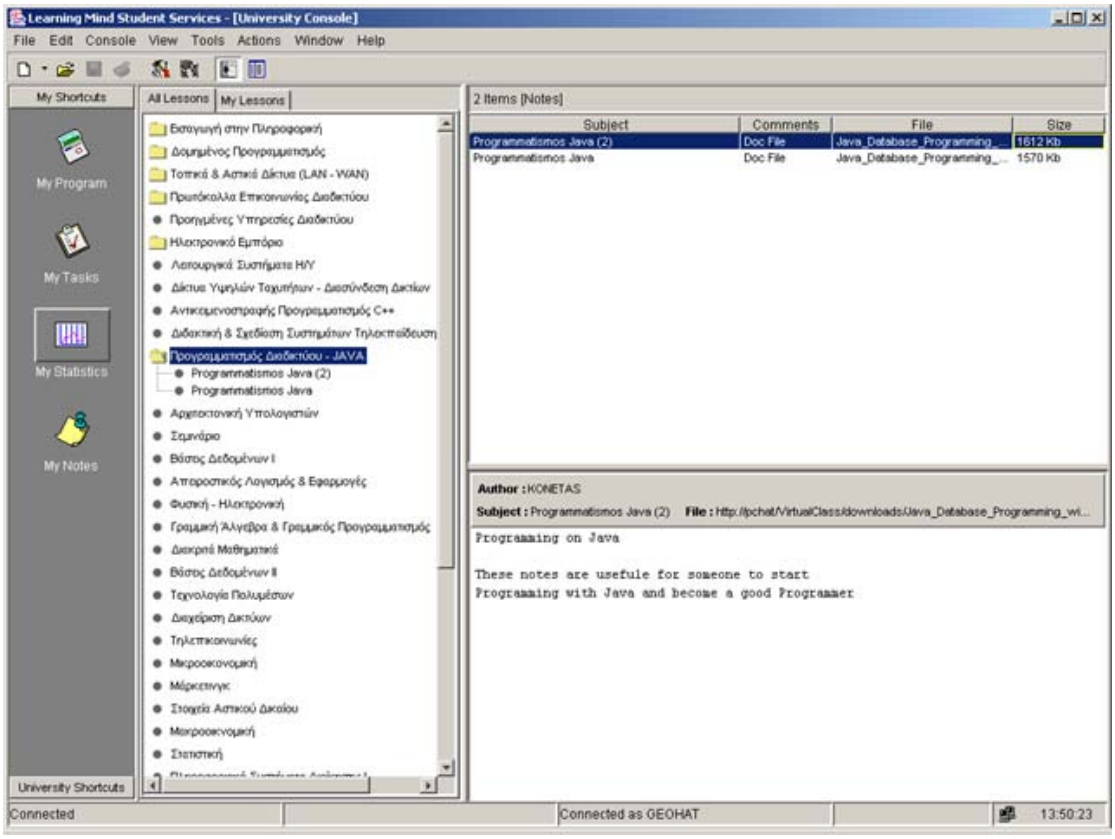
κάποιο χρήστη το καθένα από τα οποία περιέχει τα δικά του Tasks. Μαθητές και Καθηγητές μπορούν να ενημερώνονται από αυτό το παράθυρο για τις υποχρεώσεις των μαθημάτων τους και να δημιουργούν καινούριες.



Εικόνα 4.2 Κονσόλα Διαχείρισης των Tasks στο Learning Class

- Notes (Σημειώσεις). Τα Notes είναι σημειώσεις, κείμενα και αρχεία τα οποία μπορούν να αποτελέσουν την διδακτική ύλη ενός Μαθήματος. Notes δημιουργεί μόνο ένας καθηγητής για κάποιο μάθημα προς διάθεση των Μαθητών ενώ ο Μαθητής μπορεί μόνο να διαβάζει σημειώσεις, κείμενα και να κατεβάσει (Download) αρχεία για ένα συγκεκριμένο Μάθημα.

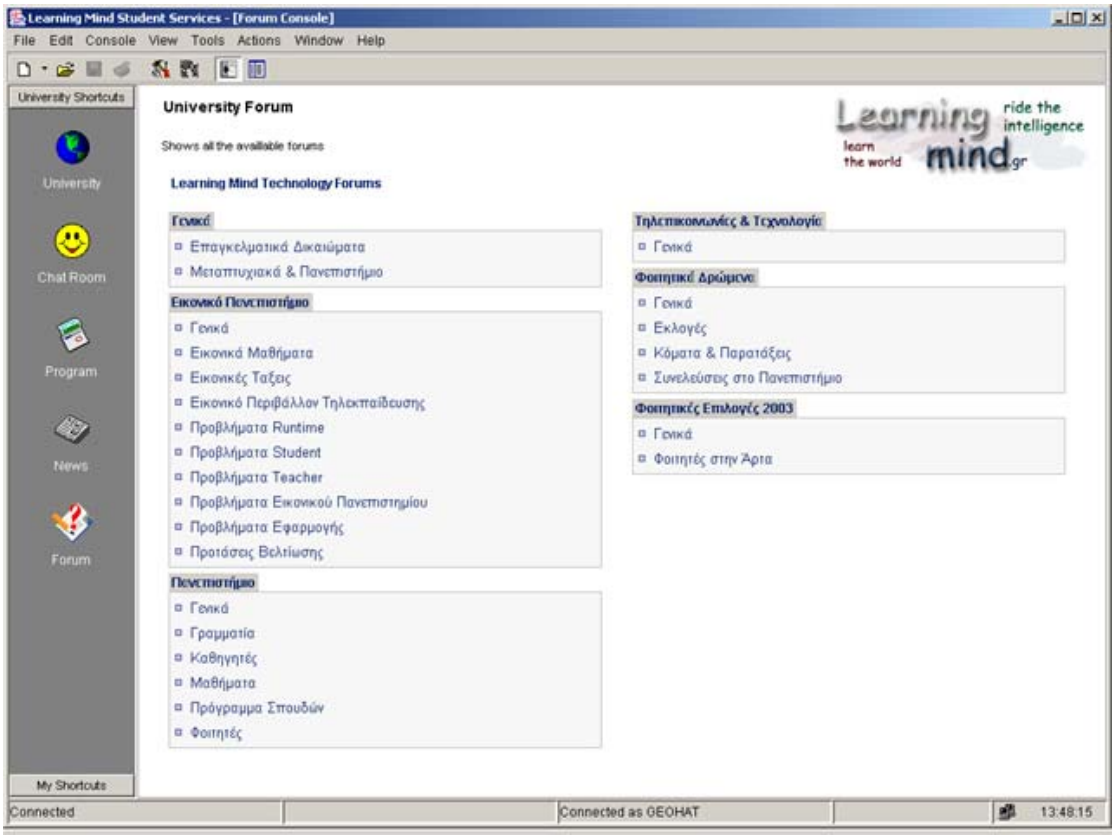
  Στην εικόνα 4.3 φαίνεται η κονσόλα διαχείρισης των Notes. Στο παράθυρο φαίνονται τα μαθήματα του Προγράμματος Σπουδών του Εικονικού Πανεπιστημίου το καθένα από τα οποία περιέχει τις δικές του σημειώσεις (Notes). Ένα Note αποτελείται από τον Τίτλο του, το κείμενό του και άλλα στοιχεία ενώ μπορεί να περιέχει και μία αναφορά σε κάποιο αρχείο το οποίο βρίσκεται στο Server διαθέσιμο σε όποιον ζητήσει να το κατεβάσει.



Εικόνα 4.3 Κονσόλα Διαχείρισης των Notes στο Learning Class

- Forum (Ομάδες Συζητήσεων). Ομάδες Συζητήσεων είναι ένας χώρος όπου μπορούν να τεθούν ερωτήματα και να αναπτυχθούν θέματα συζητήσεων μεταξύ χρηστών του Συστήματος. Πρόσβαση στις Ομάδες Συζητήσεων έχουν και Μαθητές και Καθηγητές.

  Στην εικόνα 4.4 φαίνεται η κονσόλα διαχείρισης των Forums. Στο παράθυρο φαίνονται διάφορες θεματικές ενότητες μέσα στις οποίες Μαθητές και Καθηγητές μπορούν να συζητούν θέματα που τους απασχολούν.

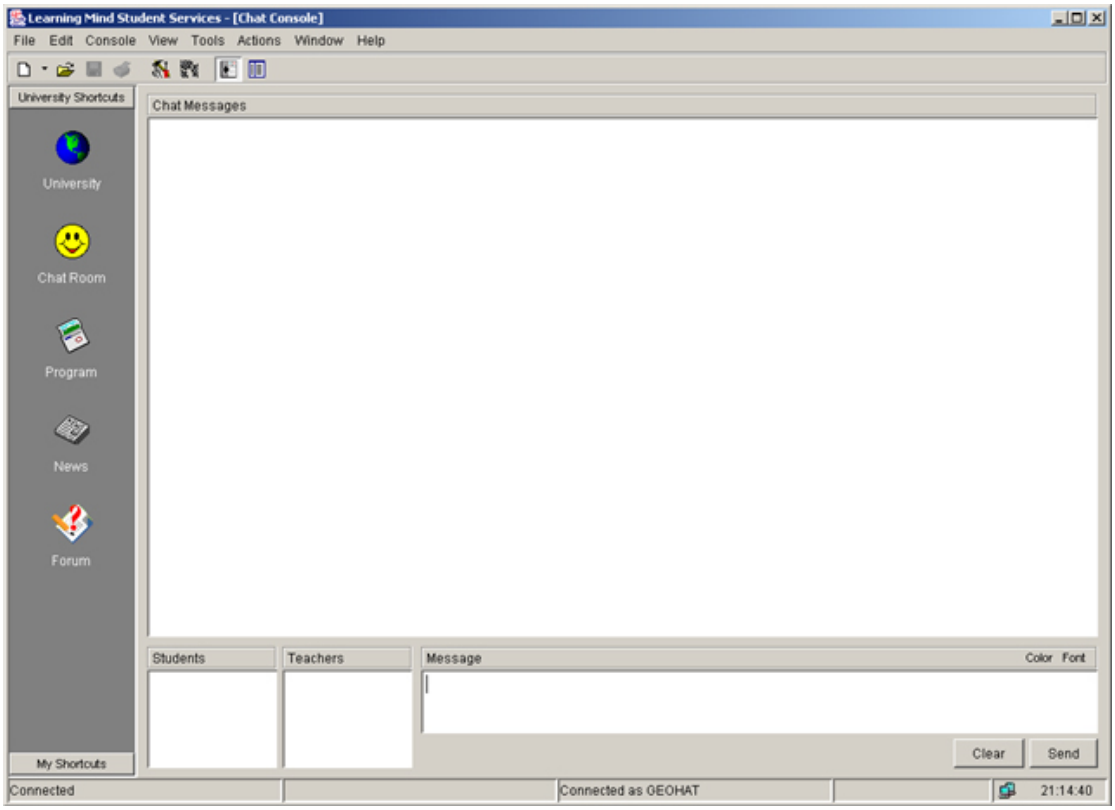


Εικόνα 4.4 Κονσόλα Διαχείρισης των Forums στο Learning Class

Εργαλεία Σύγχρονη Εκπαίδευση :

- Chat (Συζητήσεις). Το Chat είναι ευρέως διαδεδομένο εργαλείο συνδιάσκεψης μέσω Internet. Μπορεί να χρησιμοποιηθεί για να διεκπεραιώσει τις απαιτήσεις μίας τάξης ενός Μαθήματος. Καθηγητής και Μαθητές μπορούν να χρησιμοποιήσουν το Chat για online επικοινωνία μεταξύ τους.

  Στην εικόνα 4.5 φαίνεται η κονσόλα διαχείρισης των Chat. Στην ουσία το παράθυρο που βλέπουμε αποτελεί μία τάξη (Virtual Class) η οποία χρησιμοποιεί τις ελάχιστες δυνατότητες Τηλεδιάσκεψης, το Chat. Απαραίτητα στοιχεία που πρέπει να φαίνονται μέσα από την κονσόλα είναι οι συνδεδεμένοι χρήστες (Καθηγητές, Μαθητές) με τους οποίους συνδιασκέπτεται ο χρήστης κάθε στιγμή.



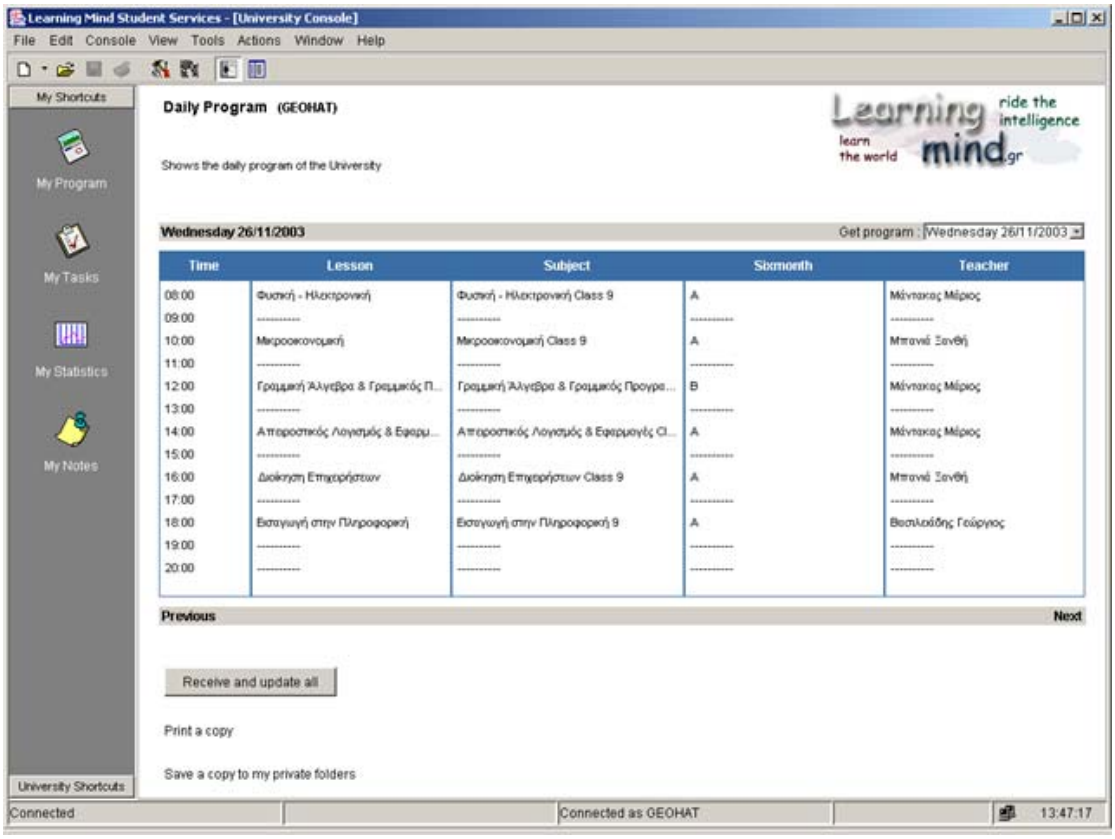
Εικόνα 4.5 Κονσόλα Διαχείρισης του Chat στο Learning Class

Άλλα Εργαλεία :

Το Learning Class παρέχει μία μεγάλη πληθώρα άλλων εργαλείων τα οποία χρησιμοποιούνται για την παροχή πληροφοριών και υπηρεσιών στους χρήστες του Συστήματος. Τέτοιες πληροφορίες είναι :

- Daily Program (Ημερήσιο Πρόγραμμα). Το Ημερήσιο Πρόγραμμα του Εικονικού Πανεπιστημίου, του Μαθητή και του Καθηγητή.
- Weekly Program (Εβδομαδιαίο Πρόγραμμα). Το Εβδομαδιαίο Πρόγραμμα του Εικονικού Πανεπιστημίου, του Μαθητή και του Καθηγητή.

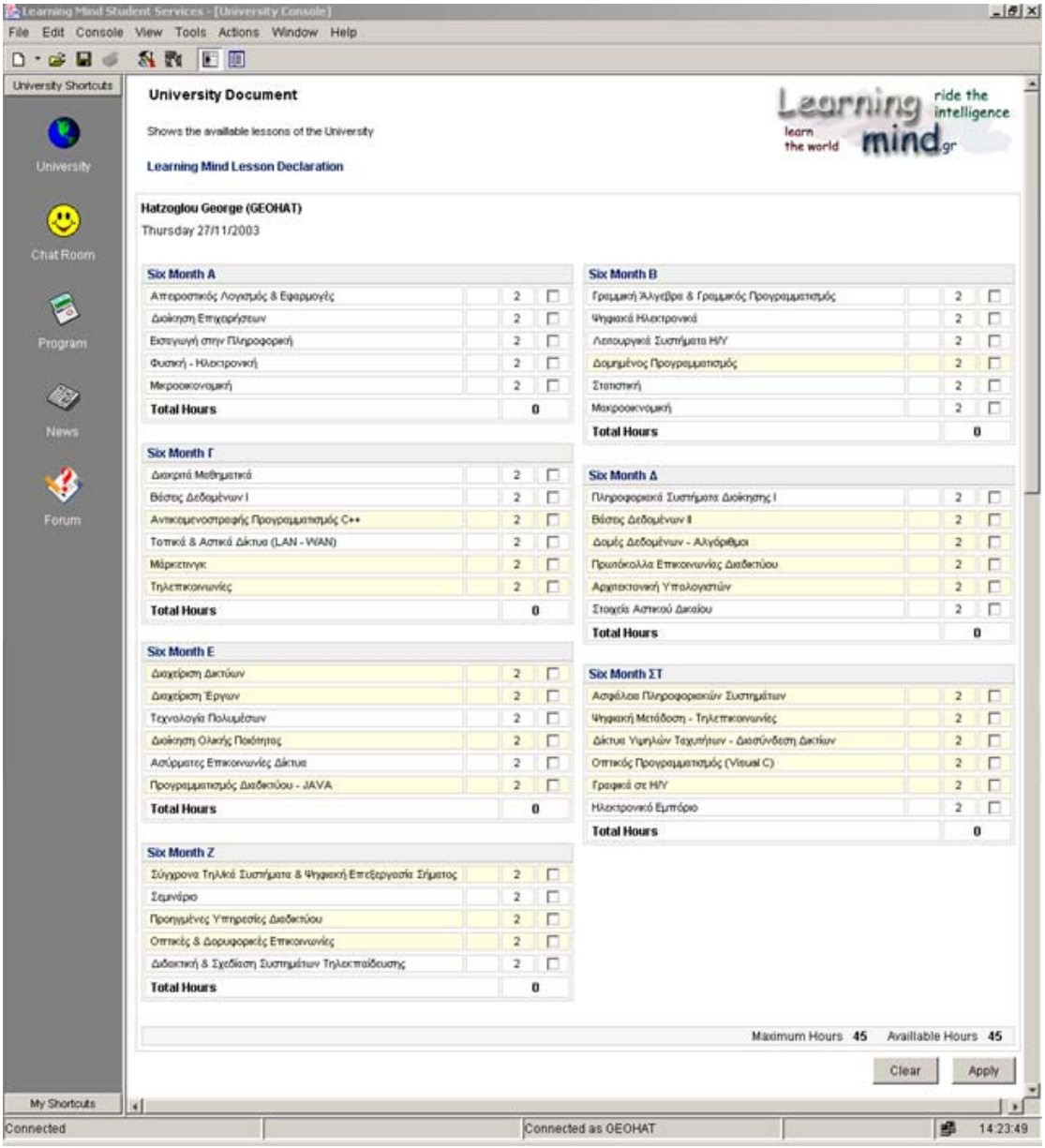
  Στην εικόνα 4.6 βλέπουμε την κονσόλα διαχείρισης των Προγραμμάτων. Η κονσόλα διαχειρίζεται το τρέχον Εβδομαδιαίο Πρόγραμμα ενώ η εμφάνιση γίνεται ανά Ημερήσιο Πρόγραμμα για ευκολία στο διάβασμα. Ο κάθε χρήστης μπορεί να μεταφέρεται στη κάθε ημέρα της εβδομάδας με τις ανάλογες επιλογές που του παρέχονται.



Εικόνα 4.6 Κονσόλα Διαχείρισης των Προγραμμάτων στο Learning Class

- Studies Program (Πρόγραμμα Σπουδών). Το Πρόγραμμα Σπουδών του Εικονικού Πανεπιστημίου.
- Lessons Declaration (Δήλωση Μαθημάτων) Αναφέρεται στο ηλεκτρονικό έγγραφο που συμπληρώνει ο Μαθητής για την δήλωση των Μαθημάτων του στην αρχή κάθε εκπαιδευτικής περιόδου.

  Στην εικόνα 4.7 βλέπουμε την κονσόλα διαχείρισης Εγγράφων (Documents). Η Δήλωση Μαθημάτων είναι το πιο σημαντικό έγγραφο ενός Εικονικού Πανεπιστημίου αφού επιτρέπει στην ουσία κάποιο Μαθητή να εγγραφεί σε μια Εκπαιδευτική Περίοδο. Στο παράθυρο φαίνεται μία Δήλωση Μαθημάτων στην οποία αναγράφονται όλα τα μαθήματα του Προγράμματος Σπουδών του Εικονικού Πανεπιστημίου με τις διδακτικές ώρες του καθενός. Μιας και η Τηλεκπαίδευση στο Σύστημά μας βασίζεται κυρίως σε εργαλεία ασύγχρονης εκπαίδευσης ο ρόλος των διδακτικών ωρών του κάθε μαθήματος είναι καθαρά περιοριστικός στο σύνολο των μαθημάτων το οποίο μπορεί να δηλώσει ο κάθε μαθητής σε μία εκπαιδευτική περίοδο. Η Δήλωση Μαθημάτων μπορεί να χρησιμοποιηθεί μόνο κατά τη διάρκεια των εγγραφών των Μαθητών.



Εικόνα 4.7 Κονσόλα Διαχείρισης Εγγράφων (Δήλωση Μαθημάτων) στο Learning Class

4.5 Κανόνες για Αποδοτική Διεξαγωγή Τηλεκπαίδευσης με το Learning Class

Βασικοί κανόνες σύμφωνα με τους οποίους λειτουργεί η Τηλεκπαίδευση στο Learning Class στηριγμένο σε ένα συμβατικό Πανεπιστήμιο είναι :

- Η διάρκεια σπουδών στο Learning Class είναι ανάλογη ενός Πανεπιστημίου χωρισμένη σε περιόδους εξαμήνων.
- Κάθε Τηλεμάθημα υπάγεται σε συγκεκριμένη περίοδο εξαμήνου.
- Κάθε Τηλεμάθημα παραδίδεται από συγκεκριμένο καθηγητή.
- Κάθε Τηλεμάθημα παραδίδεται με μορφή σημειώσεων και εργασιών.

- Η εξέταση σε κάθε Τηλεμάθημα γίνεται με μορφή τεστ τα οποία έχουν συγκεκριμένο χρόνο παράδοσης.
- Κάθε Καθηγητής είναι υποχρεωμένος να παραδίδει σημειώσεις για το Τηλεμαθημά του.
- Κάθε Καθηγητής καθορίζει την ροή του Τηλεμαθήματος με την δημοσίευση ανακοινώσεων προς τους Μαθητές.
- Κάθε Καθηγητής είναι υποχρεωμένος να μπαίνει online στο Σύστημα την ώρα που έχει δηλωθεί το Μάθημά του στο Πρόγραμμα του εξαμήνου για να επικοινωνεί με τους Μαθητές του Τηλεμαθήματος.
- Κάθε Καθηγητής είναι υποχρεωμένος να διαθέτει Forum για το κάθε Μάθημα που παραδίδει.
- Κάθε Καθηγητής είναι υποχρεωμένος να ενημερώνει τους μαθητές για τις περιόδους εξέτασης.
- Κάθε Καθηγητής είναι υποχρεωμένος να ενημερώνει τους Μαθητές του για την πρόοδο τους με δημοσίευση ανάλογων ανακοινώσεων.
- Κάθε Μαθητής παρακολουθεί τα Τηλεμαθήματα σε περιόδους εξαμήνων.
- Κάθε Μαθητής είναι ελεύθερος να επιλέγει τα Τηλεμαθήματα τα οποία θα παρακολουθήσει σε κάθε περίοδο.
- Κάθε Μαθητής είναι υποχρεωμένος να παραδίδει την δήλωση μαθημάτων του σε συγκεκριμένη χρονική περίοδο.
- Κάθε Μαθητής είναι υποχρεωμένος να υπακούει στους κανόνες και στους όρους τους οποίους ο Καθηγητής θέτει στο Μάθημά του.
- Κάθε Μαθητής έχει την δυνατότητα να θέτει ερωτήματα και απορίες για το κάθε μάθημα στο ανάλογο forum.
- Κάθε Μαθητής έχει την δυνατότητα να επικοινωνεί online με τον Καθηγητή ενός συγκεκριμένου Μαθήματος την ώρα που αυτό αναγράφεται στο Πρόγραμμα Εξαμήνου.
- Η ανάθεση των Τηλεμαθημάτων τους Καθηγητές γίνεται από τον Διαχειριστή του Συστήματος.
- Η εγγραφή των Μαθητών στο Learning Class γίνεται από τον Διαχειριστή του Συστήματος.

Κεφάλαιο 5^{ο} : Λειτουργικός Σχεδιασμός Συστήματος

5.1 Λειτουργικός Σχεδιασμός Συστήματος Τηλεκπαίδευσης (Virtual Class)

Στην περιγραφή του Συστήματος Τηλεκπαίδευσης αναφέραμε πως εξαρχής θέλαμε να υλοποιήσουμε ένα Σύστημα που να συνδυάζει ιδιότητες και χαρακτηριστικά και των δύο Μοντέλων Τηλεκπαίδευσης. Στην ουσία αυτό που θέλαμε να πετύχουμε ήταν να κατασκευάσουμε ένα Σύστημα Παροχής Υπηρεσιών Τηλεκπαίδευσης το οποίο να μπορεί να χρησιμοποιηθεί για οποιαδήποτε ανάγκη Τηλεκπαίδευσης. Χαρακτηριστικά αναφέρουμε δύο περιπτώσεις :

- Ένα Πανεπιστήμιο θέλει να κάνει χρήση του διαδικτύου για μετάδοση γνώσεων μέσω μιας σειράς Κύκλων Τηλεμαθημάτων.
- Ένα Πανεπιστήμιο θέλει να κάνει χρήση του διαδικτύου για μετάδοση γνώσεων μέσω μιας σειράς Σεμιναρίων.

Η απαίτηση αυτή να στηριχτεί ένα Σύστημα Τηλεκπαίδευσης πάνω σε ένα νέο Μοντέλο Τηλεκπαίδευσης το οποίο να συνδυάζει βασικές ιδιότητες των δύο Μοντέλων Εικονικού – Δυνητικού και Ανοιχτού Πανεπιστημίου προϋποθέτει καλό σχεδιασμό και ανάλυση. Η διαδικασία συνδυασμού των δύο Μοντέλων Τηλεκπαίδευσης σε ένα μπορεί να δημιουργήσει τον κίνδυνο να σχεδιάσουμε ένα Εικονικό Εκπαιδευτικό Ίδρυμα στο Διαδίκτυο. Ίσως εκ’ πρώτης όψεως, να φαίνεται ότι αυτό είναι ακριβώς αυτό που θέλουμε να πετύχουμε, άλλα υπάρχει μία σημαντική διαφορά που πρέπει να τονίσουμε. Το Σύστημα μας θα πρέπει να χρησιμοποιεί όρους, έννοιες και τη δομή ενός συμβατικού Εκπαιδευτικού Ιδρύματος για να ανταποκρίνεται και να λειτουργεί καλύτερα ως ένα Εικονικό Εκπαιδευτικό Ίδρυμα αλλά δεν πρέπει να είναι δεσμευμένο μόνο στις λειτουργίες ενός Πανεπιστημίου στο Διαδίκτυο. Θα πρέπει να είναι αρκετά δυναμικό και ευέλικτο ώστε να αντιπροσωπεύει κάθε είδους Σύστημα Τηλεκπαίδευσης το οποίο θέλουμε να υλοποιήσουμε. Εικονικού Πανεπιστημίου ή Ανοιχτού Πανεπιστημίου. Ανοιχτής Εκπαίδευσης μέσω σεμιναρίων ή Δρομολογημένης Εκπαίδευσης μέσω μίας σειράς Τηλεμαθημάτων. Με απαιτήσεις από τους μαθητές ή χωρίς απαιτήσεις.

Πώς θα μπορούσε να φτιαχτεί ένα Σύστημα Τηλεκπαίδευσης τόσο αυστηρό ώστε να μπορεί να παίξει το ρόλο ενός Εκπαιδευτικού Ιδρύματος στο Διαδίκτυο και ταυτόχρονα τόσο αφαιρετικό ώστε να μπορεί να χρησιμοποιηθεί ως ένα εργαλείο παρακολούθησης προαιρετικών σεμιναρίων;

Τη λύση σε ένα τέτοιο πρόβλημα την δίνει ο αντικειμενοστραφής προγραμματισμός. Κάθε συστατικό ενός τέτοιου συστήματος είναι και ένα ξεχωριστό αντικείμενο, ανεξάρτητο και αυτόνομο. Το κάθε συστατικό - αντικείμενο είναι δομημένο από συγκεκριμένες ιδιότητες και λειτουργίες αυστηρώς καθορισμένες ώστε να έχει την ευελιξία να χρησιμοποιείται σε κάθε γενικό σχεδιάγραμμα και σε κάθε ειδική απαίτησή ενός Συστήματος Τηλεκπαίδευσης.

5.2 Χρήστες του Συστήματος Τηλεκπαίδευσης

Με τον όρο χρήστη εννοούμε κάθε επισκέπτη στο Σύστημα. Οι επισκέπτες σε ένα Σύστημα Τηλεκπαίδευσης μπορούν να κατηγοριοποιηθούν ανάλογα με την ιδιότητά τους, το σκοπό της επίσκεψης τους και των απαιτήσεων που έχουν από το σύστημα αυτό καθ' αυτό. Κύριοι επισκέπτες του συστήματός μας μιας και μιλάμε για Σύστημα Τηλεκπαίδευσης είναι οι Μαθητές και οι Καθηγητές οι οποίοι συνδέονται στο Εικονικό Πανεπιστήμιο για λόγους Τηλεκπαίδευσης. Η ανάγκη όμως να ξεπεράσουμε κάποια λογικά και προγραμματιστικά προβλήματα που πηγάζουν από αυτό τον απλό διαχωρισμό μας οδηγεί σε μία νέα κατηγοριοποίηση πιο αυστηρή και δομημένη :

- Ο Απλός Χρήστης του Συστήματος (User)
- Ο Μαθητής του Εικονικού Πανεπιστημίου (Student – Virtual Student)
- Ο Καθηγητής του Εικονικού Πανεπιστημίου (Teacher – Virtual Teacher)
- Ο Διαχειριστής του Συστήματος Τηλεκπαίδευσης (Administrator – Virtual Administrator)
- Ο Εισβολέας του Συστήματος (Hacker)

Με τον διαχωρισμό αυτόν των χρηστών μπορούμε να πούμε ότι πιάνουμε όλο το εύρος των πιθανών επισκεπτών. Αυτούς που ενδιαφέρουν το Σύστημα όπως Students και Teachers και αυτούς που παίρνουν αδιάφοροι ή χρειάζονται περισσότερης προσοχής όπως οι (Hackers).

Η χρήση του όρου Hacker γίνεται για να διαχωρίσουμε όλους εκείνους τους χρήστες που δεν ενδιαφέρουν το σύστημα μας και είναι πιθανό να το βλάψουν. Η υποψία και μόνο ότι τέτοιοι χρήστες μπορεί να προσπαθήσουν να εισβάλουν στο Εικονικό Πανεπιστήμιο μας προτρέπει να δώσουμε ιδιαίτερη προσοχή στο θέμα της ασφάλειας του Συστήματος (Security). Αυτό είναι ένα θέμα που θα εκτιμηθεί σε επόμενο κεφάλαιο.

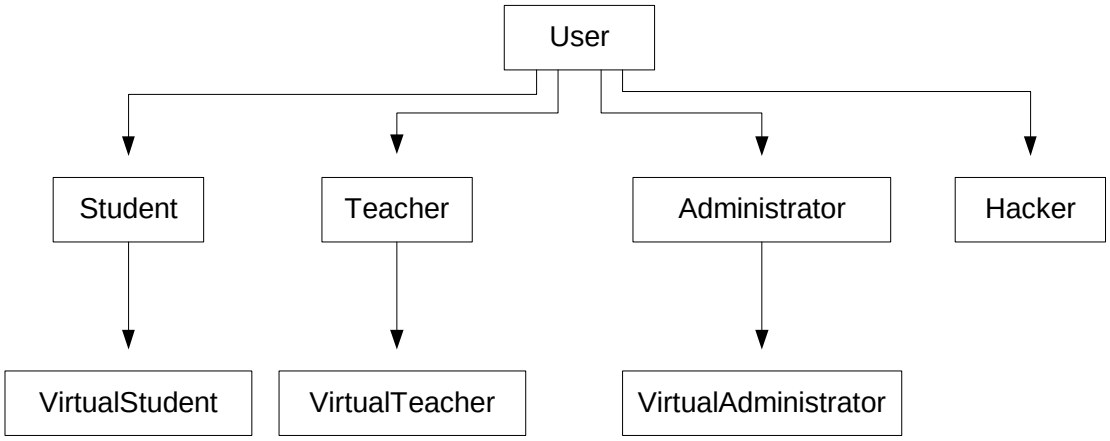
Όπως είναι φανερό, η ύπαρξη διαφορετικών επισκεπτών στο Πανεπιστήμιο απαιτεί διαφοροποίηση του συστήματος και των λειτουργιών του ανάλογα με τον κάθε χρήστη. Έτσι, θα λέγαμε πως ένα Σύστημα Τηλεκπαίδευσης θα πρέπει να είναι ικανό να προσαρμόζεται ανάλογα με τις απαιτήσεις ή τις πιθανές απαιτήσεις που μπορεί να έχει ο κάθε επισκέπτης ως προς την ιδιότητα του. Αντίθετα ο κάθε επισκέπτης θα πρέπει να υπακούει σε κανόνες και περιορισμούς που η ίδια του ιδιότητα του προσδίδει στο Σύστημα. Έτσι ένας Student δεν μπορεί παρά μόνο να δει την Βαθμολογική του κατάσταση σε κάποιο μάθημα και όχι να την επεξεργαστεί. Αντίθετα ένας Teacher μπορεί μόνο να επεξεργάζεται μια κατάσταση βαθμολογιών πολλών μαθητών ενώ δεν μπορεί να βαθμολογηθεί ο ίδιος σε κάποιο μάθημα.

Αν και ο διαχωρισμός που κάναμε πιο πάνω περιορίζει προγραμματιστικά το κάθε αντικείμενο στις συγκεκριμένες λειτουργίες του, κρίνεται επιτακτική η ανάγκη να περιορίσουμε τους χρήστες και με άλλους τρόπους. Μια ευρέως χρησιμοποιημένη λογική είναι να περιορίσουμε κάθε χρήστη με δικαιώματα (Permissions). Permission είναι μια ‘Άδεια’ ώστε κάποιος να κάνει κάτι. Αυτό σημαίνει ότι μπορούμε να δίνουμε Άδειες στους χρήστες ώστε να μπορούν να χρησιμοποιούνε κάποιες υπηρεσίες του Συστήματος ενώ άλλες όχι. Είναι πολύ σημαντικό να πούμε πως η χρήση των Permissions μας δίνει τη δυνατότητα να περιορίζουμε χρήστες της ίδιας ιδιότητας. Έτσι δύο Students ενώ είναι και οι δύο μαθητές στο Πανεπιστήμιο μπορούν να έχουν διαφορετικά δικαιώματα στις υπηρεσίες του Συστήματος ανάλογα τον αριθμό των μαθημάτων που έχουν περάσει ή τη βαθμολογική τους κατάσταση ή ανάλογος άλλων παραγόντων.

Προς το παρόν μιας και η υλοποίηση του συστήματος θα γίνει σε αντικειμενοστραφής γλώσσα προγραμματισμού μπορούμε να πούμε πως η κάθε οντότητα χρήστη που δημιουργήσαμε αποτελεί και ένα ξεχωριστό αντικείμενο. Το κάθε ένα αντικείμενο περιορίζεται στις συγκεκριμένα στοιχεία που αφορούν την ιδιότητα του ως χρήστη. Οι λειτουργίες του κάθε αντικειμένου αφορούνε μόνο την

ανάκτηση και επεξεργασία των στοιχείων του και όχι χρήση υπηρεσιών του συστήματος. Θα φαινότανε λογικό αν σε κάθε χρήστη δίναμε τη δυνατότητα Login (Σύνδεση στο Πανεπιστήμιο), κάτι τέτοιο όμως θα δέσμευε το ίδιο το αντικείμενο χρήστη σε αυτήν την υπηρεσία και δε θα άφηνε περιθώρια ανεξαρτητοποίησης. Η υπηρεσία Login που αναφέραμε και γενικά κάθε άλλη υπηρεσία που έχει στη διάθεση του ο χρήστης δεν είναι μέρος της δομής του αλλά ενέργειες που κάποιος άλλος τις κάνει διαθέσιμες σε αυτόν. Ποίος παρεμβάλλεται ανάμεσα σε χρήστη και υπηρεσίες θα το εξετάσουμε στη συνέχεια.

Η χρήση των αντικειμένων στη δομή που προσπαθούμε να χτίσουμε μας δίνει την δυνατότητα να χρησιμοποιήσουμε την κληρονομικότητα ως ένα σημαντικό κλειδί στο σχεδιασμό του συστήματος. Έτσι μπορούμε να δημιουργήσουμε ένα κληρονομικό δέντρο χρηστών μίας και η κάθε ιδιότητα επισκέπτη εμπεριέχει κοινά στοιχεία με τις άλλες ιδιότητες (Εικόνα 5.1).



Εικόνα 5.1 Αναπαράσταση του Δέντρου Χρηστών του Συστήματος Τηλεκπαίδευσης

5.2.1 Απλός Χρήστης (User)

Το αντικείμενο User αναφέρεται στον απλό χρήστη του Συστήματος. Το όνομα του φανερώνει τις περιορισμένες του δυνατότητες και γενικά του δίνει την ιδιότητα του περαστικού, κάποιου δηλαδή που απλώς θέλει να αντλήσει πληροφορίες από το σύστημα χωρίς να έχει την δυνατότητα να παρέμβει στην Τηλεκπαίδευση αφού δεν είναι μαθητής στο Εικονικό Πανεπιστήμιο. Οι πληροφορίες που μπορεί να αντλήσει ο User είναι αυτές που θα θεωρήσουμε εμείς γενικές και ελεύθερες να έχει πρόσβαση να βλέπει.

Στην συνέχεια θα χρησιμοποιούμε τον όρο User ως αντικείμενο το οποίο έχει περιορισμένες και γενικές ιδιότητες και λειτουργίες. Ιδιότητές User θεωρούμε κάθε πληροφορία χρήσιμη και απαραίτητη, η οποία βοηθάει στην ταυτοποίηση του χρήστη, την αναγνώρισή του από το Σύστημα και στην διευκόλυνση της Επικοινωνίας του με το Εικονικό Πανεπιστήμιο.

Ο User είναι το πρώτο στοιχείο ενός δέντρου κληρονομικότητας το οποίο κληροδοτεί τις ιδιότητες του στα υπόλοιπα αντικείμενα (χρήστες). Τέτοιες ιδιότητες είναι το συνθηματικό του για την σύνδεση του με το Σύστημα και η IP του με την οποία κάνει την σύνδεση.

5.2.2 Μαθητής (Student – Virtual Student)

Είναι ίσως το πιο σημαντικό στοιχείο του Συστήματός μας αφού ο όρος Τηλεκπαίδευση και κάθε Εφαρμογή που προσφέρει υπηρεσίες Τηλεκπαίδευσης δημιουργήθηκαν από την ανάγκη του Μαθητή να αντλεί γνώσεις και πληροφορίες από άλλες πηγές Εκπαίδευσης απομακρυσμένα. Η ανάγκη αυτή δημιουργεί αυτόματα την απαίτηση, το Σύστημα μας να έχει ως κύριο στόχο και άξονα τον ίδιο τον Μαθητή και την υπακοή στους νόμους και τους όρους που αυτός ορίζει στην Τηλεκπαίδευση.

Στην συνέχεια θα χρησιμοποιούμε τον όρο Student ως αντικείμενο το οποίο έχει συγκεκριμένες ιδιότητες και λειτουργίες. Ένα Student είναι ένας κλώνος User, δηλαδή πολλά από τα στοιχεία του είναι στοιχεία που τα κληρονομεί από την ιδιότητά του ως User. Πρόσθετα στοιχεία του Student είναι όλες εκείνες οι πληροφορίες που τον ταυτοποιούνε ως Μαθητή του Εικονικού Πανεπιστημίου και όλες εκείνες οι πληροφορίες απαραίτητες για την διεξαγωγή της διαδικασίας της Τηλεκπαίδευσης. Αναφορικά ορισμένες από αυτές είναι η βαθμολογία του, το πρόγραμμα μαθημάτων του, το απουσιολόγιό του.

Ένας Student θα πρέπει να είναι εξουσιοδοτημένος να επεξεργάζεται και να διαγράφει μόνο πληροφορίες οι οποίες αναφέρονται σε αυτόν και μόνο. Αντίθετα θα πρέπει να έχει την δυνατότητα να αντλεί και να βλέπει κάθε είδους πληροφορία για όποιο θέμα ή άτομο αυτός θέλει. Η απαίτηση αυτή μας οδηγεί στο να εισαγάγουμε μια καινούρια έννοια, την έννοια του Virtual Student. Ένας Virtual Student δεν είναι παρά ένα Μαθητής κληρονόμος του Student με περισσότερα στοιχεία. Η δημιουργία δύο αντικειμένων Student εξυπηρετεί ένα και μόνο σκοπό, την δυνατότητα να έχουμε

κάποιες πληροφορίες του μαθητή διαθέσιμες και ορατές σε άλλους χρήστες ενώ κάποιες άλλες όχι. Ένας Virtual Student είναι και ένας Student και ένας User. Αυτό σημαίνει ότι ανάλογα με την ιδιότητα που είναι συνδεδεμένος ο Μαθητής στο Σύστημα θα έχει και τα αντίστοιχα δικαιώματα πάνω σε πληροφορίες δικές του και άλλων.

5.2.3 Καθηγητής (Teacher – Virtual Teacher)

Ο Καθηγητής είναι αναπόφευκτα κι αυτός ένα πολύ σημαντικό συστατικό ενός Συστήματος Τηλεκπαίδευσης. Ο Καθηγητής είναι υπεύθυνος για την διεκπεραίωση των απαιτήσεων ενός Τηλεμαθήματος και είναι αυτός που ορίζει τον τρόπο της διεξαγωγής του. Είναι εξουσιοδοτημένος να ορίζει τους δικούς του κανόνες και όρους στο μάθημα και στους μαθητές του υπακούοντας πάντα στους περιορισμούς του συστήματος και στις δυνατότητες που του προσφέρει. Ο Καθηγητής είναι υποχρεωμένος να διαθέτει κάποιας μορφής σημειώσεων στους μαθητές για το μάθημα του, να κάνει εξέταση και αξιολόγηση μαθητών και να βοηθάει τους μαθητές για την καλύτερη κατανόηση της ύλης.

Στην συνέχεια θα χρησιμοποιούμε τον όρο Teacher ως ένα αντικείμενο που θα εξυπηρετεί τις ανάγκες ενός καθηγητή. Ένα Teacher είναι κληρονόμος ενός User. Δηλαδή ένας Teacher εκτός από την ιδιότητά του ως Καθηγητής περικλείει και την ιδιότητα του απλού χρήστη που του επιτρέπει να συνδέεται και να επικοινωνεί με το πανεπιστήμιο. Κύρια στοιχεία του είναι όλες εκείνες οι πληροφορίες που τον ταυτοποιούνε ως Καθηγητή στο Εικονικό Πανεπιστήμιο και όλες εκείνες απαραίτητες για την διεξαγωγή της Τηλεκπαίδευσης.

Το αντικείμενο Virtual Teacher είναι όπως και στην περίπτωση του Μαθητή ένας Καθηγητής κληρονόμος του Teacher με πρόσθετες πληροφορίες. Ένας Virtual Teacher είναι και ένας Teacher και ένας User. Έτσι σε μία συγκεκριμένη στιγμή ο Καθηγητής θα έχει τα ανάλογα δικαιώματα της ιδιότητας που είναι συνδεδεμένος με το σύστημα.

5.2.4 Διαχειριστής (Administrator)

Απαραίτητη προϋπόθεση στην καλή λειτουργία ενός Συστήματος Τηλεκπαίδευσης και γενικά κάθε Κατανεμημένου Συστήματος είναι η ύπαρξη κάποιου ατόμου εξουσιοδοτημένου να επιβλέπει το Σύστημα και να ελέγχει τη ροή και τη λειτουργία του. Τον ρόλο αυτό τον παίζουν συνήθως οι Διαχειριστές (Administrators). Τα άτομα αυτά είναι υπεύθυνα για την καλή λειτουργία των Συστημάτων, για την διόρθωση βλαβών, επίλυση λαθών και προσφορά οποιαδήποτε βοήθειας στους υπόλοιπους χρήστες του συστήματος.

Όπως βλέπουμε ένας διαχειριστής είναι επιφορτισμένος με πλήθος υποχρεώσεων πάνω στο Σύστημα που επιβλέπει. Για την επίτευξη των υποχρεώσεων αυτών θα πρέπει να του παραχωρηθούν και τα ανάλογα δικαιώματα (Permissions) διαχείρισης από το Σύστημα έτσι ώστε να μπορεί να επεμβαίνει όταν υπάρχει ανάγκη.

Στο δικό μας σύστημα τον ρόλο του Διαχειριστή τον παίζει ο Administrator που είναι ένα αντικείμενο χρήστη κληρονόμος του User. Ο Administrator εμπεριέχει στοιχεία απαραίτητα για να τον αναγνωρίζει το Σύστημα ως διαχειριστή καθώς και όλες τις πληροφορίες του χρήστη. Περιέχει επίσης πρόσθετα δικαιώματα (Permissions) που του επιτρέπουν να έχει περισσότερες δυνατότητες, ίσως και διαφοροποιημένες από ένα άλλο χρήστη στο Σύστημα.

Εισάγεται επίσης η έννοια του Virtual Administrator ο οποίος είναι ένας κληρονόμος Administrator με πρόσθετες πληροφορίες και δικαιώματα.

5.2.5 Εισβολέας (Hacker)

Ο όρος Εισβολέας αναφέρεται γενικά σε χρήστες οι οποίοι συνδέονται σε Συστήματα με σκοπό να τα βλάψουν ή να έχουν παράνομη πρόσβαση στις πληροφορίες τους. Εισβολή σε Κατανεμημένα Συστήματα Πληροφορικής θεωρείται κάθε προσπάθεια επικοινωνίας με τους διακομιστές και κάθε προσπάθεια ανάκτησης πληροφοριών του Συστήματος από μη εξουσιοδοτημένους χρήστες.

Η ανάγκη πρόληψης για αποφυγή τέτοιων εισβολών σε ένα Εικονικό Πανεπιστήμιο μας προτρέπει να ορίσουμε ως ένα νέο είδος χρήστη το Hacker. Αυτός δεν είναι παρά ένας απλός User στον οποίο πρέπει να κόψουμε κάθε δικαίωμα πρόσβασης στα δεδομένα του Πανεπιστημίου. Ένα Σύστημα Τηλεκπαίδευσης θα πρέπει να είναι ικανό να αναγνωρίζει τους χρήστες αυτούς και να τους περιορίζει ή

ακόμα να είναι προγραμματισμένο αν φέρει ειδικής μεταχείρισης αυτών των χρηστών για εξασφάλιση της ασφάλειας των αποθηκευμένων ή διακινούμενων πληροφοριών.

Το αντικείμενο Hacker είναι κληρονόμος του User με επιπρόσθετα στοιχεία τα οποία αφορούν ένα εισβολέα.

5.3 Εικονικό Πανεπιστήμιο (University)

Η έννοια του Εικονικού Πανεπιστημίου θα μπορούσαμε να πούμε πως είναι η αντίστοιχη ενός Πανεπιστημιακού Ιδρύματος με την μόνη διαφορά ότι η έδρα του πρώτου και χώρος δραστηριοποίησης του είναι το Διαδίκτυο. Ένα Εικονικό Πανεπιστήμιο δεν είναι παρά ένας εικονικός χώρος συγκέντρωση μαθητών και καθηγητών. Μέσα σ' αυτό το χώρο μαθητές και καθηγητές μπορούν να ανακτούν πληροφορίες για τις σπουδές τους, να βλέπουν ανακοινώσεις, να θέτουν προβλήματα και ερωτήσεις, να ανακτούν έγγραφα, σημειώσεις και βιβλία και γενικά να συνδιασκέπτονται με κάθε τρόπο. Στην ουσία το University θα παίζει το ρόλο του Εικονικού κτιριακού χώρου στον οποίο συγκεντρώνονται όλοι οι χρήστες του Πανεπιστημίου ανεξάρτητα την ιδιότητά τους. Όπως σε ένα Πανεπιστήμιο όπου ο οποιοσδήποτε έχει το δικαίωμα να εισέλθει, να διαβάσει ανακοινώσεις και να συζητήσει με τους άλλους παρευρισκόμενους στο χώρο του Πανεπιστημίου την ίδια δυνατότητα πρέπει να έχει και ένα Εικονικό Πανεπιστήμιο.

Στο Σύστημα μας το University θα είναι ένα αντικείμενο υπεύθυνο για την διαχείριση των Πληροφοριών του Πανεπιστημίου και θα ασχολείται με την διάθεσή αυτών στους χρήστες που τις ζητάνε. Επίσης θα είναι υπεύθυνο για τη διαχείριση των χρηστών και θα τους προσφέρει όλες τις υπηρεσίες του Εικονικού Πανεπιστημίου.

5.4 Γραμματεία (Secretariat)

Ο όρος Γραμματεία σε ένα Πανεπιστήμιο αναφέρεται κυρίως σε ένα κτιριακό χώρο ή σε ένα σύνολο ατόμων τα οποία ασχολούνται με την παροχή πληροφοριών, εγγράφων, σημειώσεων και βιβλίων σε μαθητές και καθηγητές. Σε ένα Εικονικό Πανεπιστήμιο ο όρος Γραμματεία μπορεί να χρησιμοποιηθεί σαν μια οντότητα η οποία θα είναι υπεύθυνη για την παροχή ανάλογων πληροφοριών σε χρήστες του Συστήματος Τηλεκπαίδευσης σε ηλεκτρονική μορφή.

Παραπάνω είχαμε αναφέρει πως με τη διαχείριση των πληροφοριών του Συστήματος θα ασχολείται το αντικείμενο University. Αξίζει εδώ να σημειωθεί πως

το University θα είναι υπεύθυνο για την παροχή υπηρεσιών και πληροφοριών στους χρήστες. Όλα τα δεδομένα του Πανεπιστημίου θα τα διαχειρίζεται το Secretariat και θα τα κάνει διαθέσιμα στους χρήστες μέσω του University. Από εδώ προκύπτει πως ένα αντικείμενο University θα εμπεριέχει μέσα στις ιδιότητες του και ένα αντικείμενο Secretariat. Κάτι ανάλογο με όλα τα Εκπαιδευτικά Ιδρύματα όπου σε κάθε ένα από αυτά υπάρχει και μία Γραμματεία.

Έτσι στο Σύστημα μας το Secretariat θα είναι ένα αντικείμενο υπεύθυνο για την διαχείριση των δεδομένων του Πανεπιστημίου. Όπως είναι λογικό όλα αυτά τα δεδομένα και οι πληροφορίες πρέπει να αποθηκεύονται σε κάποια μορφή Βάσης Δεδομένων. Συνεπακόλουθο αυτού είναι ότι όταν θα λέμε πως το Secretariat θα κάνει διαχείριση των δεδομένων θα μιλάμε για διαχείριση της συγκεκριμένης Βάσης Δεδομένων. Κανένα άλλο αντικείμενο δεν πρέπει να έχει πρόσβαση ή τουλάχιστον πρόσβαση χωρίς την άδεια της Γραμματείας στη Βάση. Αυτό εξυπηρετεί και κάποιο είδος ασφάλειας στα δεδομένα μας αφού ο μόνος που θα μπορεί να επεξεργαστεί απευθείας τα στοιχεία του Πανεπιστημίου είναι ένα μόνο αντικείμενο, το Secretariat. Άλλωστε, αυτού του είδους η λογική ακολουθεί και τα πρότυπα ενός καλά δομημένου προγράμματος όπου η κάθε οντότητα εξυπηρετεί συγκεκριμένες λειτουργίες και κάθε λειτουργία εξυπηρετείται από συγκεκριμένη "Υπηρεσία".

5.5 Πρόγραμμα Σπουδών (StudiesProgram)

Ένα πρόγραμμα σπουδών σε ένα Ίδρυμα γενικά είναι ένα σύνολο μαθημάτων τα οποία θεωρούνται απαραίτητα για την απόκτηση του Πτυχίου του Εκπαιδευτικού Ιδρύματος. Τα μαθήματα είναι ταξινομημένα με κάποια ορισμένη σειρά στις διάφορες εκπαιδευτικές περιόδους έτσι ώστε ο κάθε μαθητευόμενος να ακολουθεί μια λογική συνέχεια στην εκπαίδευσή του.

Στο Σύστημά μας ένα Πρόγραμμα Σπουδών θα ορίζεται από το αντικείμενο StudiesProgram και το οποίο θα περιέχει όλα τα μαθήματα με τη ορισμένη τους σειρά στην εκπαίδευση, και ένα όνομα για το συγκεκριμένο Πρόγραμμα. Όπως θα εξετάσουμε παρακάτω μία εκπαιδευτική περίοδος (Sixmonth) του Πανεπιστημίου περιέχει ένα σύνολο μαθημάτων. Έτσι ένα StudiesProgram θα ορίζεται από ένα σύνολο εκπαιδευτικών περιόδων οι οποίες δημιουργούνε το σύνολο των μαθημάτων του Προγράμματος Σπουδών από το άθροισμα των μαθημάτων των περιόδων.

5.6 Εξάμηνο (Sixmonth)

Το εξάμηνο είναι μια ευρέως χρήσημοποιημένη έννοια σε Εκπαιδευτικά Ιδρύματα και Πανεπιστήμια. Αναφέρεται σε μία χρονική περίοδο της εκπαιδευτικής χρονιάς χωρισμένης σε εξάμηνα. Μέσα σε αυτή τη χρονική περίοδο πραγματοποιείται ένας κύκλος εκπαίδευσης στης οποίας το τέλος πραγματοποιούνται οι καθορισμένες εξετάσεις για την αξιολόγηση των μαθητευομένων.

Μιας και μιλάμε για ένα Σύστημα Τηλεκπαίδευση ενός Εικονικού Πανεπιστημίου είναι φανερή η ανάγκη της χρήσης αυτού του όρου στο σχεδιασμό μας. Έτσι θα εισαγάγουμε ένα νέο αντικείμενο με το όνομα Sixmonth και το οποίο θα αντιπροσωπεύει μια χρονική περίοδο της Εκπαίδευσης. Θα πρέπει να τονίσουμε πως αν και το όνομα του αντικειμένου που προορίσαμε αναφέρεται σε ένα στατικό αριθμό 6 μηνών η χρήση του είναι αρκετά δυναμική. Δηλαδή το Sixmonth θα αποτελεί μία περίοδο σπουδών με δυνατότητα να οριστεί η χρονική της διάρκεια ανάλογα με τις απαιτήσεις του Συστήματος. Έστω πρόγραμμα σπουδών ενός Εικονικού Πανεπιστημίου το οποίο αποτελείται από 7 εξάμηνα. Αυτό συνεπάγεται ότι το Σύστημα του πανεπιστημίου θα διαχειρίζεται 7 Sixmonth το κάθε ένα από τα οποία θα αναφέρεται σε ένα συγκεκριμένο εξάμηνο από τα 7.

Ιδιότητες ενός Sixmonth θα είναι όπως προαναφέραμε η χρονική του διάρκεια μία λίστα των μαθημάτων πού υπάγονται στη συγκεκριμένοι περίοδο και το όνομα της περιόδου.

5.7 Βασικοί Όροι Μάθημα – Τάξη – Βιβλίο

Πριν συνεχίσουμε είναι σημαντικό να κάνουμε ένα διαχωρισμό τριών εννοιών που στην καθομιλουμένη πολλές φορές δημιουργείται μια σύγχυση θέλοντας να αναφερθούμε σε αυτές. Έστω ένα Μάθημα με το όνομα ‘Αντικειμενοστραφής Προγραμματισμός’ και έστω η φράση ‘Πάω για μάθημα (Αντικειμενοστραφής Προγραμματισμός)’. Μπορεί στον προφορικό μας λόγο αυτή η φράση να μας εξυπηρετεί και να είναι ικανή ώστε να μας κάνει να συνεννοηθούμε με τον συνομιλητή μας στην κυριολεξία όμως από την φράση αυτή πηγάζουνε τρεις σημασίες.

- Πάω να παρακολουθήσω το Μάθημα του Προγράμματος με το όνομα Αντικειμενοστραφής Προγραμματισμός.

- Πάω να παρακολουθήσω το συγκεκριμένο μάθημα το οποίο παρουσιάζεται στην τάξη με όνομα Αντικειμενοστραφής Προγραμματισμός.
- Πάω να παρακολουθήσω το μάθημα τού οποίου η ύλη είναι από το βιβλίο Αντικειμενοστραφής Προγραμματισμός.

Όπως είναι φανερό η σύγχυση αυτή δεν εξυπηρετεί καθόλου το Σύστημα μας. Μίας και το Μάθημα είναι σημαντικός παράγοντας σε ένα Σύστημα Τηλεκπαίδευσης είναι σημαντικό να θέσουμε τρεις διαφορετικές οντότητες που οι οποίες να ορίζουν τις τρεις αυτές έννοιες.

Μάθημα (Lesson) θα εννοούμε το μάθημα το οποίο είναι εγγεγραμμένο στο πρόγραμμα σπουδών του Πανεπιστημίου και το οποίο είναι διαθέσιμο στους μαθητές να το παρακολουθήσουν. Το Lesson θα το παρουσιάζει συγκεκριμένος Καθηγητής – Καθηγητές και θα έχει συγκεκριμένη ύλη (βιβλία, σημειώσεις).

Τάξη (Class) θα εννοούμε τον εικονικό χώρο όπου θα συνδιασκέπτονται μαθητές και καθηγητές για την διεξαγωγή συγκεκριμένου μαθήματος.

Σημείωση (Note) θα εννοούμε ένα κείμενο ή κείμενα τα οποία αποτελούνε την ύλη για κάποιο μάθημα.

5.8 Μάθημα (Lesson)

Όπως προαναφέραμε με τον όρο Μάθημα θα εννοούμε κάθε μάθημα του προγράμματος σπουδών του Εικονικού Πανεπιστημίου. Ένα Lesson δεν είναι τίποτε άλλο παρά ένα αντικείμενο το οποίο περιέχει πληροφορίες για το συγκεκριμένο μάθημα, όπως ο τίτλος του οι διδακτικές μονάδες του, το είδος του μαθήματος και οι διδακτικές του ώρες. Μέσα στις ιδιότητες του απαραίτητο είναι να υπάρχουν λίστες με τους εγγεγραμμένους μαθητές στο μάθημα και τους καθηγητές που το παρουσιάζουν.

5.9 Τάξη (Class – Virtual Class)

Η Τάξη θα μπορούσαμε να πούμε πως είναι το πιο σημαντικό κομμάτι του Συστήματος αφού εδώ συγκεντρώνονται όλα τα εργαλεία για την παροχή της Τηλεκπαίδευσης. Όπως είπαμε η Τάξη είναι ένας εικονικός χώρος τηλεδιάσκεψης χρηστών (Μαθητών, Καθηγητών). Χρησιμοποιείται σε περιπτώσεις σύγχρονης εκπαίδευσης αφού απαιτεί συνύπαρξη των χρηστών σε συγκεκριμένες στιγμές και online επικοινωνία μεταξύ τους. Η επικοινωνία των χρηστών πρέπει να γίνεται με

όλους τους διαθέσιμους τρόπους οι οποίοι χρησιμοποιούνται σήμερα στο Διαδίκτυο όπως Chat, Whiteboard, Video και Audio Conference. Επίσης θα πρέπει να υπάρχει δυνατότητα ανταλλαγής σημειώσεων και αρχείων μεταξύ μαθητών και καθηγητών για διευκόλυνση της εκπαίδευσης.

Εδώ πρέπει να τονίσουμε την ανάγκη ύπαρξης ενός αντικειμένου το οποίο πρέπει να κρατάει όλα τα στοιχεία και τις πληροφορίες μίας Τάξης ενώ η παροχή των υπηρεσιών επικοινωνίας και τηλεδιάσκεψης πρέπει να ελέγχεται από ανεξάρτητο αντικείμενο. Έτσι θα εισάγουμε μία νέα οντότητα το Virtual Class το οποίο θα παίζει το ρόλο της Τάξης και θα παρέχει όλα τα εργαλεία Τηλεκπαίδευσης ενώ το Class θα είναι το αντικείμενο που θα κρατάει τα στοιχεία κάθε Τάξης. Τέτοια στοιχεία του Class θα είναι το μάθημα το οποίο θα παρουσιαστεί, η ημερομηνία και η ώρα του μαθήματος, το θέμα συζήτησης του μαθήματος.

5.10 Σημείωση (Note)

Σημείωση αποτελεί κάθε βιβλίο, κείμενο ή κείμενα τα οποία έχει ορίσει ο καθηγητής ως ύλη του μαθήματος του. Οι σημειώσεις θα δημιουργούνται από καθηγητές και θα διανέμονται σε οποιοδήποτε ενδιαφέρεται για το συγκεκριμένο μάθημα. Παράλληλα με τον καθηγητή τη δυνατότητα δημιουργίας σημειώσεων θα πρέπει να την έχει και ο μαθητής με μόνη διαφορά ότι οι σημειώσεις των μαθητών είναι προσωπικές ενώ αυτές των καθηγητών είναι διαθέσιμες για όλους.

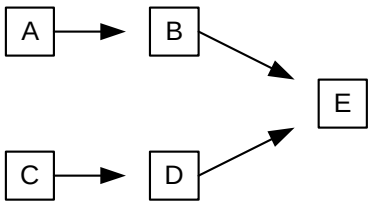
Απαραίτητα στοιχεία μίας σημείωσης είναι ο τίτλος της, ένα απλό κείμενό, ο συγγραφέας της, και μία τοποθεσία στο Internet από όπου μπορεί να γίνει download του ηλεκτρονικού βιβλίου. Στην ουσία δηλαδή ένα Note είναι αναφορά σε ένα ηλεκτρονικό βιβλίο και όχι το ίδιο το βιβλίο.

5.11 Αλυσίδα (Linked Lesson)

Γενικά ο όρος αλυσίδα χρησιμοποιείται για να δηλώσει πως κάποια μαθήματα ακολουθούν μια λογική σειρά στον τρόπο παράδοσης της ύλης τους. Δηλαδή μία αλυσίδα είναι μια σειρά μαθημάτων την οποία πρέπει να ακολουθούν οι μαθητές για να κατανοούν καλύτερα τις θεματικές ενότητες που αναπτύσσονται σε κάθε ένα από τα μαθήματα αυτά. Οι αλυσίδες στα Συμβατικά Ιδρύματα τηρούνται υποχρεωτικά. Στο Εικονικό Πανεπιστήμιο που εμείς υλοποιούμε αφήνουμε τις αλυσίδες ελεύθερες και η χρήση τους χρησιμοποιείται καθαρά για πληροφοριακούς λόγους. Για να

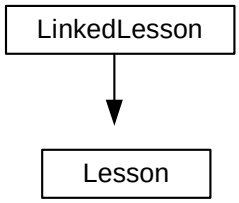
γνωρίζει ο κάθε μαθητής δηλαδή την σειρά που είναι καλό να ακολουθήσει στην εκπαίδευση του για να κατανοήσει καλύτερα την ύλη των μαθημάτων.

Στο Σύστημά μας ένα LinkedLesson είναι ένα αντικείμενο το οποίο περιέχει μία λίστα με τα προαπαιτούμενα μαθήματα που πρέπει να παρακολουθήσει κάποιος πριν το αναφερόμενο. Έστω η αλυσίδα της εικόνας 5.2. Τότε το Linked Lesson E θα περιέχει μία λίστα με δύο Linked Lesson το B και το D. Ενώ το B θα περιέχει στη λίστα του το A και το D το C. Έτσι ανά πάσα στιγμή μπορούμε να γνωρίζουμε ολόκληρη την αλυσίδα ή αλυσίδες ενός μαθήματος βάση της λειτουργίας των συνδεδεμένων λιστών.



Εικόνα 5.2 Παράδειγμα Αλυσίδας Μαθημάτων

Το LinkedLesson είναι ένας κληροδότης του Lesson. Δηλαδή ένα Lesson αποτελεί και μία αλυσίδα η οποία μπορεί να περιέχει προαπαιτούμενα μαθήματα ενώ μπορεί και όχι (Εικόνα 5.3).



Εικόνα 5.3 Κληρονομικό Δέντρο Αλυσίδας Μαθημάτων

5.12 Προγράμματα (Programs)

Η έννοια του Προγράμματος προσδιορίζει ένα σύνολο Μαθημάτων τα οποία είναι προγραμματισμένα να διεκπεραιωθούν μέσα στη χρονική περίοδο στην οποία αναφέρεται το Πρόγραμμα. Η χρήση του βασίζεται στην ανάγκη του κάθε χρήστη του Εικονικού Πανεπιστημίου να γνωρίζει ανά πάσα στιγμή τα μαθήματα του και τις ώρες που είναι προσδιορισμένα να γίνουν. Από αυτό φαίνεται πως βασικά στοιχεία που πρέπει να φαίνονται μέσα από ένα Πρόγραμμα είναι η χρονική περίοδος στην

οποία αναφέρεται, τα μαθήματα της περιόδου και η χρονική στιγμή πραγματοποίησης του κάθε μαθήματος. Εκτός από αυτά, πρόσθετα στοιχεία μπορούν να φαίνονται μέσα σε ένα Πρόγραμμα όπως ο Καθηγητής που παραδίδει το κάθε μάθημα και η εκπαιδευτική περίοδος στην οποία υπάγεται το κάθε μάθημα.

Η χρονική περίοδος ενός Προγράμματος μπορεί να είναι ημερήσια ή εβδομαδιαία. Δηλαδή ένα Πρόγραμμα μπορεί να αναφέρεται στα μαθήματα μίας ημέρας ή μιας εβδομάδας. Επειδή όμως όπως έχουμε προαναφέρει θέλουμε να σχεδιάσουμε ένα Σύστημα όσο το δυνατό πιο δυναμικό και ευέλικτο έτσι και ένα Πρόγραμμα θα πρέπει να είναι το ίδιο δυναμικό και ευέλικτο αποδεσμευμένο από το χρόνο. Έτσι θα ορίσουμε τρία αντικείμενα τα οποία θα εξυπηρετούν τις παραπάνω απαιτήσεις.

- Ημερήσιο Πρόγραμμα (DailyProgram). Το Ημερήσιο Πρόγραμμα είναι σαφώς δεσμευμένο από το χρόνο ο οποίος αναφέρεται σε μία Εκπαιδευτική ημέρα.
- Εβδομαδιαίο Πρόγραμμα (WeeklyProgram). Το Εβδομαδιαίο Πρόγραμμα αναφέρεται σε μία Εκπαιδευτική εβδομάδα. Είναι ένα σύνολο 7 Ημερησίων Προγραμμάτων τα οποία προσδιορίζουν ολόκληρο το Εβδομαδιαίο Πρόγραμμα. Είναι και αυτό δεσμευμένο απ το χρόνο.
- Γενικό Πρόγραμμα (Program). Το Γενικό Πρόγραμμα είναι ένα δυναμικό πρόγραμμα στο οποίο μπορούμε να δηλώσουμε εμείς τη χρονική διάρκεια του προγράμματος σε ημέρες. Δηλαδή το Γενικό Πρόγραμμα είναι ένα σύνολο Ημερησίων Προγραμμάτων ανάλογο του αριθμού των ημερών που θα δηλώσουμε.

Τα δύο Προγράμματα Ημερήσιο, Εβδομαδιαίο χρησιμοποιούνται για τις ανάγκες ενός Εικονικού Πανεπιστημίου όπου το Πρόγραμμα επαναλαμβάνεται σε εβδομάδες μέσα σε μία Εκπαιδευτική Περίοδο (Εξάμηνο). Το Γενικό Πρόγραμμα είναι σχεδιασμένο για οποιαδήποτε χρήση αφού η χρονική περίοδος δεν προσδιορίζεται εξαρχής αλλά μπορεί να οριστεί ανάλογα με τις ανάγκες μας.

5.13 Έγγραφα (Documents)

Με τον όρο έγγραφο θα εννοούμε κάθε πληροφορία που μπορεί να ζητηθεί και να ανακτηθεί από τη Γραμματεία ενός Εικονικού Πανεπιστημίου σε ηλεκτρονική μορφή. Τέτοια έγγραφα μπορεί να είναι τα γνωστά έγγραφα ενός Πραγματικού

Πανεπιστημίου όπως η Δήλωση Μαθημάτων ή άλλα έγγραφα αναγκαία για την διεξαγωγή της Εκπαίδευσης μέσα από το Διαδίκτυο, όπως μία λίστα με τους συνδεδεμένους χρήστες μέσα σε μία Εικονική Τάξη. Βασικές ιδιότητες των εγγράφων είναι η ικανότητα τους να μεταφέρονται μέσα στο Διαδίκτυο και να εκτυπώνονται από εκτυπωτές των Υπολογιστικών Συστημάτων που θα χειρίζονται οι χρήστες του Εικονικού Πανεπιστημίου. Τις ιδιότητες αυτές θα πρέπει να τις κληρονομεί κάθε αντικείμενο το οποίο θα ικανοποιεί τις απαιτήσεις ενός Document.

5.14 Δήλωση Μαθημάτων (Lessons Declaration)

Με τον όρο Δήλωση Μαθημάτων εννοούμε ένα Ηλεκτρονικό Έγγραφο μέσω του οποίου ένας Μαθητής ενός Εικονικού Πανεπιστημίου μπορεί να δηλώσει τα μαθήματα που θέλει να παρακολουθήσει μέσα σε μία Εκπαιδευτική Περίοδο. Ο όρος πηγάζει από τα Συμβατικά Πανεπιστήμια και τα στοιχεία που περιέχει είναι σχεδόν παρόμοια με ένα ανάλογο έγγραφο.

Μια Δήλωση Μαθημάτων πρέπει να περιέχει όλες τις Εκπαιδευτικές Περιόδους του Εικονικού Πανεπιστημίου με τα μαθήματα που τις απαρτίζουν. Στην ουσία μία δήλωση αποτελείται από το Πρόγραμμα Σπουδών του Πανεπιστημίου μέσα στο οποίο είναι δηλωμένα, Εκπαιδευτικές Περίοδοι και Μαθήματα. Απαραίτητη προϋπόθεση μίας δήλωσης είναι να υπάρχουν οι διδακτικές ώρες του κάθε μαθήματος. σε περίπτωση Σύγχρονης Εκπαίδευσης. Οι διδακτικές μονάδες των μαθημάτων, στην περίπτωση που οι σπουδές στο Εικονικό Πανεπιστήμιο οδηγούν σε κάποιας μορφής πιστοποίησης γνώσεων και αξιολόγησης των μαθητευομένων στα Τηλεμαθήματα.

Ένα αντικείμενο Δήλωσης Μαθημάτων θα πρέπει να περιέχει πρόσθετες μεταβλητές για το κάθε μάθημα (flags) με τις οποίες θα μπορεί να εξακριβωθεί αν το κάθε μάθημα έχει δηλωθεί ή όχι. Πρέπει να υπάρχει όριο για το αριθμό μαθημάτων τον οποίο μπορεί να χρησιμοποιήσει κάποιος μαθητής ο οποίος να είναι δυναμικός έτσι ώστε να μπορεί να οριστεί ανάλογα με την πολιτική του Εικονικού Πανεπιστημίου.

Η Δήλωση Μαθημάτων είναι ηλεκτρονικό έγγραφο (Document) το οποίο διαχειρίζεται η Γραμματεία του Εικονικού Πανεπιστημίου. Συνεπώς το LessonsDeclaration είναι ένας κληρονόμος του Document το οποίο κληρονομεί τις βασικές ιδιότητες των εγγράφων.

Κεφάλαιο 6ο : Υλοποίηση Συστήματος Τηλεκπαίδευσης (Virtual Class)

6.1 Υλοποίηση Συστήματος Τηλεκπαίδευσης (Virtual Class)

Θεωρούμε τα δύο αντικείμενα Γραμματεία (Secretariat), Εικονικό Πανεπιστήμιο (University) ως τα πιο σημαντικά και ως τη βάση στο σχεδιασμό ενός Συστήματος Τηλεκπαίδευσης τον οποίο αναλύσαμε στα προηγούμενα κεφάλαια. Στη συνέχεια θα εξηγήσουμε το πώς υλοποιούνται τα δύο αυτά αντικείμενα και θα δούμε τον τρόπο με τον οποίο οι απομακρυσμένοι χρήστες του Εικονικού Πανεπιστημίου έρχονται σε επικοινωνία με αυτά προς χρήση των υπηρεσιών που προσφέρουν.

6.2 Υλοποίηση Αντικείμένου Secretariat

Το αντικείμενο Secretariat όπως είπαμε είναι το αντικείμενο το οποίο θα κάνει διαχείριση της βάσης δεδομένων του Εικονικού Πανεπιστημίου και θα παρέχει όλες τις αποθηκευμένες πληροφορίες προς τα έξω. Στην ουσία μέσα στο Secretariat θα υλοποιούνται ερωτήσεις Queries προς στη Βάση Δεδομένων για την ανάκτηση των δεδομένων. Ας υποθέσουμε πως η Γραμματεία υλοποιεί συνάρτηση για το διάβασμα όλων των μαθημάτων του Πανεπιστημίου. Τα μαθήματα αυτά θα επιστρέφονται σε μία λίστα η οποία θα περιέχει τα μαθήματα με όλες τις πληροφορίες τους.

Γραμματεία (Secretariat)

```
public class Secretariat {
  public Secretariat() {
  }
  public ListLessons GetListLessons() {
    ListLessons listLessons = new ListLessons();
    // διάβασμα από τη Βάση για ανάκτηση των Μαθημάτων και
    // γέμισμα του listLessons με τα δεδομένα.
    return (listLessons)
  }
}
```

Από τον παραπάνω κώδικα φαίνεται πως σε κάθε περίπτωση που θα χρειαστούμε τα Μαθήματα του Πανεπιστημίου αρκεί να χρησιμοποιήσουμε την συνάρτηση GetListLessons(). Όπως είναι φανερό σε κάθε χρήση της συνάρτησης γίνεται και ένα νέο διάβασμα από τη βάση. Είναι δηλαδή πιθανόν για μεγάλο χρονικό διάστημα όπου δεν θα γίνει καμία αλλαγή στα Μαθήματα του Πανεπιστημίου να γίνεται το ίδιο διάβασμα στη βάση σε κάθε αίτησή μας. Αυτό δείχνει να είναι πρόβλημα αν σκεφτούμε πως κάθε επικοινωνία με τη Βάση υφίσταται και μία καθυστέρηση χρόνου, ειδικά στην περίπτωση που η βάση είναι εγκαταστημένη σε δικό της Server απομακρυσμένα από το Σύστημα όπου τρέχει η Γραμματεία. Αν σκεφτούμε ακόμα την περίπτωση οπού ένας μεγάλος αριθμός αιτήσεων πραγματοποιηθεί την ίδια χρονική στιγμή, τότε θα έχουμε ανάλογες καθυστερήσεις στο Σύστημα με τον αριθμό των αιτήσεων. Το πρόβλημα μπορεί να λυθεί αν η συνάρτηση επιστρέφει από την μνήμη τα ίδια δεδομένα σε κάθε αίτηση και να τα ανανεώνει από τη βάση μόνο στην περίπτωση που θα πραγματοποιηθεί κάποια αλλαγή στη βάση. Ας δούμε τις αλλαγές που θα πρέπει να γίνουν στο κώδικα.

Γραμματεία (Secretariat)

```
public class Secretariat {
 private ListLessons listLessons = new ListLessons();
 public Secretariat() {
  LoadListLessons();
 }
 public ListLessons GetListLessons() {
  return (listLessons)
 }
 public synchronized void InsertLesson(Lesson newLesson) {
  // εισαγωγή του καινούριου μαθήματος () στη βάση
  LoadListLessons();      // ανανέωση των Μαθημάτων
 }
 public synchronized void LoadListLessons() {
  ListLessons newList = new ListLessons();
  // διάβασμα από τη Βάση για ανάκτηση των Μαθημάτων και γέμισμα του
  // newList με τα δεδομένα.
```

```
    listLessons = new List;
    // ανανέωση των δεδομένων
  }
}
```

Στον παραπάνω κώδικα διατηρούμε μία μεταβλητή (listLessons) η οποία θα περιέχει τα μαθήματα του Εικονικού Πανεπιστημίου. Μόλις δηλωθεί το αντικείμενο Secretariat η μεταβλητή αυτή γεμίζει με ένα διάβασμα από τη βάση (LoadListLessons). Σε κάθε κλήση της συνάρτησης GetListLessons() θα γίνεται επιστροφή της μεταβλητής που διατηρούμε στην μνήμη η οποία περιέχει τις πληροφορίες που ζητήσαμε. Αν πραγματοποιηθεί μία αλλαγή στη λίστα μαθημάτων (InsertLesson) τότε τα καινούρια δεδομένα διαβάζονται από τη Βάση και η μεταβλητή στη μνήμη μας ανανεώνεται.

Έτσι αν σε μία χρονική στιγμή γίνουν 100 κλήσεις της συνάρτησης GetListLessons θα γίνουν 100 επιστροφές της μεταβλητής από την μνήμη και η ταχύτητα εξυπηρέτησης θα είναι ανάλογη της ταχύτητας του Συστήματός μας. Στην περίπτωση που γίνει κλήση της μεθόδου InsertLesson(Lesson newLesson) τότε θα πραγματοποιηθεί 1 διάβασμα στη βάση για ανανέωση των δεδομένων και με κάθε νέα αίτηση για τα μαθήματα θα επιστρέφονται τα ανανεωμένα δεδομένα.

Το synchronized είναι μία ειδική λέξη στη Java η οποία χρησιμοποιείται σε περιπτώσεις που θέλουμε συγχρονισμό δεδομένων. Ειδικά σε περιπτώσεις Κατανεμημένων Συστημάτων όπου πολλές πηγές μπορούν να έχουν πρόσβαση στα ίδια δεδομένα την ίδια στιγμή το synchronized μας επιτρέπει να εξασφαλίζουμε ασφαλή επεξεργασία των δεδομένων και γενικά να δεσμεύουμε τιμές και μεταβλητές για όσο γίνεται η επεξεργασία τους. Έτσι αποφεύγουμε τον κίνδυνο κάποιος άλλος να προσπαθήσει επεξεργαστεί τα ίδια δεδομένα, πράγμα που μπορεί να οδηγήσει σε λάθος ή σε απώλεια πληροφοριών. Η χρήση του είναι σαν ένα είδος κλειδαριάς το οποίο κλειδώνει μια συνάρτηση μέχρι το τέλος της εκτέλεσής της. Έτσι αν σε μία χρονική στιγμή γίνουν δύο ταυτόχρονες κλήσεις της μεθόδου InsertLesson τότε μία από τις δύο αιτήσεις θα εκτελεστεί πρώτη (αυτή της οποίας η αίτηση έφτασε πρώτη στη Γραμματεία) και η άλλη μετά το τέλος της εκτέλεσης της πρώτης. Η χρήση της λέξης synchronized γίνεται για να υπάρχει συγχρονισμός στις κλήσεις των μεθόδων InsertLesson και LoadListLessons. Έτσι η εισαγωγή των μαθημάτων θα γίνει με τη

λογική σειρά που έγιναν οι αιτήσεις. Ενώ αιτήσεις για διάβασμα της μεταβλητής από την GetListLessons μπορούν να γίνονται ελεύθερα..

Έτσι μπορούμε να συνοψίσουμε τις βασικές αρχές πάνω στις οποίες θα κινείται το αντικείμενο Secretariat.

- Θα διατηρεί μεταβλητές στη μνήμη για κάθε πληροφορία την οποία θα χρησιμοποιεί.
- Θα διατηρεί μεταβλητές στη μνήμη για κάθε πληροφορία που πρέπει να διαβάζει από την Βάση Δεδομένων.
- Θα κάνει διαθέσιμες αυτές τις μεταβλητές προς τα έξω.
- Θα ανανεώνει τις τιμές των μεταβλητών όποτε και αν αυτό κρίνεται αναγκαίο.
- Θα εξασφαλίζει ότι οι τιμές των μεταβλητών είναι σωστές.
- Θα εξασφαλίζει την ασφαλή και σωστή επεξεργασία Δεδομένων.

6.3 Υλοποίηση Αντικείμένου University

Το αντικείμενο University όπως είπαμε είναι το αντικείμενο στο οποίο θα υλοποιούνται όλες οι υπηρεσίες του Εικονικού Πανεπιστημίου. Βασικές λειτουργίες του είναι να μπορεί να υλοποιεί συνδέσεις χρηστών και να κάνει διαχείριση αυτών. Απ την άλλη θα πρέπει να κάνει προσβάσιμες όλες τις πληροφορίες της Γραμματείας στους συνδεδεμένους χρήστες. Όπως και στην περίπτωση του Secretariat έτσι και στο University πολλές πληροφορίες θα κρατιούνται στη μνήμη για ταχύτερη πρόσβαση και εξυπηρέτηση των αιτήσεων.

Ας υποθέσουμε ότι το University υλοποιεί συναρτήσεις για τις συνδέσεις των χρηστών (Login), συναρτήσεις για την ανάκτηση των πληροφοριών των χρηστών και συνάρτηση για ανάκτηση του Προγράμματος Σπουδών.

Πανεπιστήμιο (University)

```
public class University {
  private ListStudent connectedStudents = new ListStudents();
  private ListTeachers connectedTeachers = new ListTeachers();
  private Secretariat secretariat = new Secretariat();
  private StudiesProgram studiesProgram = new StudiesProgram();
  public University() {
  }
```

```
public synchronized int LoginStudent(User user) {
 Student student = new Student();
 if (secretariat.IdentifyUser(user).equals("Student")) {
  // επικοινωνία με τη βάση και γέμισμα του student για τον συγκεκριμένο
  // User
  return (connectedStudents.Add(student));
 }
 else {
  return (-1);
 }
}
public synchronized int LoginTeacher(User user) {
 Teacher teacher = new Teacher();
 if (secretariat.IdentifyUser(user).equals("Teacher")) {
  // επικοινωνία με τη βάση και γέμισμα του teacher για τον
  // συγκεκριμένο User
  return (connectedTeachers.Add(teacher));
 }
 else {
  return (-1);
 }
}
public void LogoutStudent(int studentID) {
 // επικοινωνία με τη βάση για αποδέσμευση του
 // συγκεκριμένου Student
 connectedStudents.Remove(studentID);
}
public void LogoutTeacher(int teacherID) {
 //επικοινωνία με τη βάση για αποδέσμευση του συγκεκριμένου Teacher
 connectedTeachers.Remove(teacherID);
}
public ListLessons GetListLessons() {
 return (secretariat.GetListLessons());
```

```
  }
  public synchronized void LoadStudiesProgram() {
    StudiesProgram newProgram = new StudiesProgram();
    // επικοινωνία με τη βάση και τη Γραμματεία για γέμισμα του newProgram
    studiesProgram = newProgram;
  }
  public StudiesProgram GetStudiesProgram() {
    return (studiesProgram);
  }
  public ListStudents GetConnectedStudents() {
    return (connectedStudents);
  }
  public ListTeachers GetConnectedTeachers() {
    return (connectedTeachers);
  }
}
```

Οι μεταβλητές connectedStudents, connectedTeachers είναι λίστες στις οποίες κρατάμε τους συνδεδεμένους χρήστες του συστήματός μας. Κάθε φορά που ένας καινούριος επισκέπτης συνδέεται με το Πανεπιστήμιο (Login), τότε αυτός προστίθεται στους συνδεδεμένους χρήστες ανάλογα με την ιδιότητά του και παίρνει ένα αναγνωριστικό αριθμό από το σύστημα. Στην ειδική περίπτωση που δεν αναγνωριστούν τα στοιχεία κάποιου User από τη βάση του συστήματος τότε επιστρέφεται η τιμή -1 από την συνάρτηση Login και ο User πρέπει να ξαναπροσπαθήσει να συνδεθεί.

Από τη στιγμή πού θα συνδεθεί κάποιος επισκέπτης και μέχρι τη στιγμή της αποσύνδεσης του μπορεί να αντλεί πληροφορίες από το πανεπιστήμιο από τις διαθέσιμες συναρτήσεις. Στον παραπάνω κώδικα οι χρήστες μπορούν να χρησιμοποιήσουν την συνάρτηση GetStudiesProgram για την ανάκτηση του Προγράμματος Σπουδών του Πανεπιστημίου.

Η μέθοδος GetStudiesProgram όπως είναι φανερό δεν έχει συγκεκριμένες απαιτήσεις για να τρέξει αφού δεν παίρνει καμία είσοδο στην υλοποίηση της. Εδώ μπορεί να συμπεράνει κανείς πως κάποιος χρήστης ακόμα και να μην είναι

συνδεδεμένος με το Πανεπιστήμιο μπορεί να χρησιμοποιήσει την συνάρτηση και να αντλήσει τη συγκεκριμένη πληροφορία. Αυτό όπως είναι φανερό είναι λάθος αφού οι πληροφορίες πρέπει να είναι διαθέσιμες μόνο σε εξουσιοδοτημένους χρήστες. Για να λυθεί αυτό το πρόβλημα θα μπορούσαμε να διαμορφώσουμε την συνάρτηση GetStudiesProgram και κάθε άλλη συνάρτηση να παίρνει είσοδο τα στοιχεία του χρήστη που κάνει την αίτηση έτσι ώστε με ένα μικρό έλεγχο των στοιχείων το σύστημα να αποφασίζει αν θα επιστρέψει την πληροφορία ή όχι. Αν και αυτός ο τρόπος αρχικά φαίνεται σωστότερος πρέπει νωρίτερα να επισημάνουμε δύο πράγματα.

- Το University δεν αποφασίζει ποίος θα έχει πρόσβαση στις πληροφορίες του Πανεπιστημίου αλλά υλοποιεί υπηρεσίες για ελεύθερη πρόσβαση χωρίς να ενδιαφέρεται ιδιαίτερα ποιος θα τις χρησιμοποιήσει.
- Η υπηρεσία σύνδεσης χρήστη (Login) είναι μια τυπική διαδικασία αναγνώρισης επισκέπτη που επιτρέπει περαιτέρω διαχείριση και όχι σύνδεση σε υπηρεσίες του Πανεπιστημίου.

Από τα παραπάνω προκύπτει πως την διαχείριση των υπηρεσιών του πανεπιστημίου και των έλεγχο πρόσβασης στα δεδομένα θα πρέπει να γίνονται από κάποια άλλη οντότητα ενώ το University θα πρέπει να διατηρήσει την δυναμικότητα και την ελευθερία που ήδη του δώσαμε. Άλλωστε πρέπει να περιορίσουμε τις αιτήσεις μεταξύ χρηστών και Συστήματος στο ελάχιστο έτσι ώστε να μειώσουμε το φόρτο επικοινωνίας του δικτύου. Έτσι κάθε υπηρεσία του University θα πρέπει να χρησιμοποιείται μόνο όταν πρέπει (όταν ο χρήστης είναι συνδεδεμένος) και θα πρέπει πάντα να επιστρέφει μια τιμή. Το πώς γίνεται η διαχείριση των υπηρεσιών και το πώς αυτές γίνονται διαθέσιμες στους χρήστες θα το εξετάσουμε σε επόμενη ενότητα.

Έτσι μπορούμε να συνοψίσουμε τις βασικές αρχές πάνω στις οποίες θα κινείται το αντικείμενο University.

- Θα διατηρεί μεταβλητές στη μνήμη για κάθε πληροφορία την οποία θα χρησιμοποιεί.
- Θα κάνει διαθέσιμες αυτές τις μεταβλητές προς τα έξω.
- Θα ανανεώνει τις τιμές των μεταβλητών όποτε και αν αυτό κρίνεται αναγκαίο.
- Θα εξασφαλίζει ότι οι τιμές των μεταβλητών είναι σωστές.
- Θα εξασφαλίζει την ασφαλή και σωστή επεξεργασία Δεδομένων.
- Θα υλοποιεί συναρτήσεις σύνδεσης – αποσύνδεσης χρηστών.

- Θα κάνει διαχείριση των συνδεδεμένων χρηστών.
- Θα υλοποιεί συναρτήσεις (υπηρεσίες) πρόσβασης στης πληροφορίες Τηλεκπαίδευσης.

Κεφάλαιο 7ο : Client – Server Programming

7.1 Client – Server Programming

Όπως είδαμε ένα Σύστημα Τηλεκπαίδευσης είναι ένα κατανεμημένο σύστημα απομακρυσμένων υπολογιστών στους οποίους τρέχουν τα κατάλληλα προγράμματα και υπηρεσίες παρέχοντας δυνατότητες Τηλεκπαίδευσης στους χρήστες. Στο κεφάλαιο αυτό θα εξετάσουμε πώς επιτυγχάνεται η επικοινωνία του συστήματος και των απομακρυσμένων χρηστών ανάμεσα στις συνδέσεις δικτύου.

Παραπάνω αναφέραμε τις βασικές οντότητες και τα αντικείμενα πάνω στα οποία θα στηριχτεί το Σύστημα μας. Θεωρήσαμε το αντικείμενο University ως τον βασικό κορμό επικοινωνίας χρηστών και δεδομένων αλλά δεν αναφερθήκαμε καθόλου στον παράγοντα απόσταση η οποία παρεμβάλλεται ανάμεσα τους. Η εισαγωγή του όρου απόσταση στο σχεδιασμό μας και γενικά η παρεμβολή του διαδικτύου ανάμεσα στις οντότητες και υπηρεσίες του Συστήματος μας οδηγεί να μπούμε σε μια πιο ειδική μορφή σχεδιασμού και προγραμματισμού, του Δικτυακού Προγραμματισμού (Client – Server Programming).

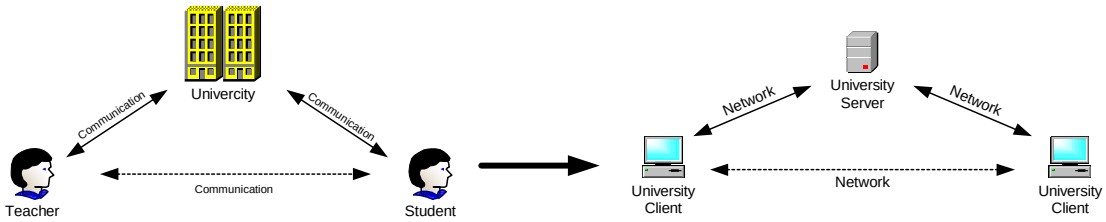
Ο Δικτυακός Προγραμματισμός υφίσταται με την συνύπαρξη δύο βασικών στοιχείων ενός δικτυακού τόπου:

- Διακομιστής (Server). Είναι κάθε υπολογιστικό σύστημα η ειδικό πρόγραμμα Διακομιστή το οποίο είναι εξουσιοδοτημένο να εξυπηρετεί τις αιτήσεις απομακρυσμένων πελατών.
- Πελάτης (Client). Είναι κάθε υπολογιστικό σύστημα ή ειδικό πρόγραμμα Πελάτη το οποίο κάνει απομακρυσμένες αιτήσεις σε εξουσιοδοτημένους Διακομιστές για συλλογή πληροφοριών.

Όπως βλέπουμε η επικοινωνία ανάμεσα σε δύο απομακρυσμένα στοιχεία ενός δικτύου γίνεται με δύο βασικές λειτουργίες

1. Αίτηση του Πελάτη στο Διακομιστή για την παροχή πληροφοριών.
2. Απάντηση του Διακομιστή στην αίτηση του πελάτη και παροχή των ανάλογων πληροφοριών.

Ας δούμε ένα σχεδιάγραμμα που μας δείχνει ένα Πανεπιστήμιο με τις βασικές Δικτυακές απαιτήσεις:



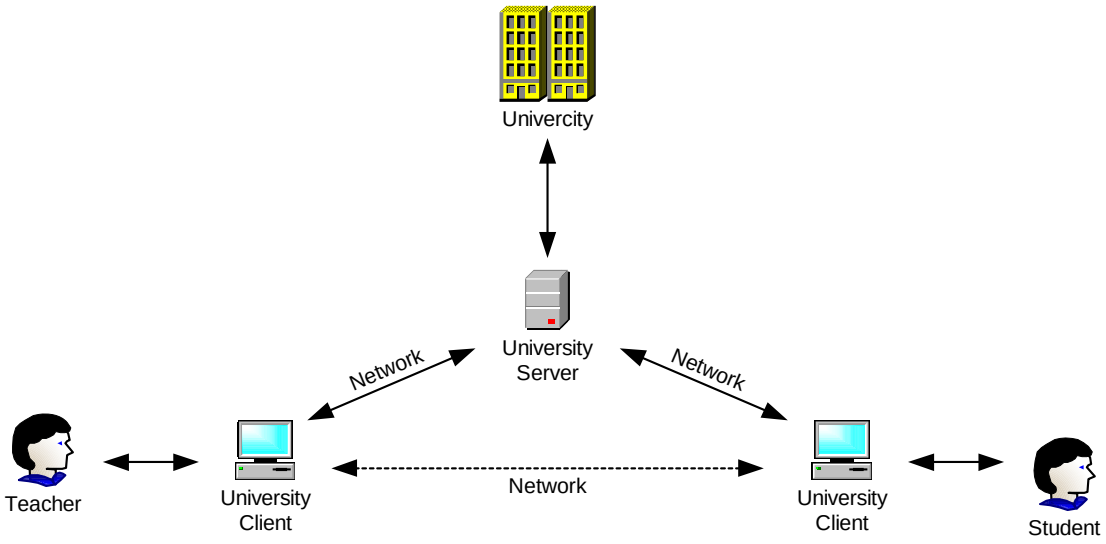
Εικόνα 7.1 Σχεδιάγραμμα με τις Βασικές Δικτυακές Απαιτήσεις του Συστήματος Τηλεκπαίδευσης

Όπως φαίνεται από την εικόνα 7.1 το University θα πρέπει να τρέχει κάτω από μία υπηρεσία Διακομιστή (University Server) ο οποίος θα εξυπηρετεί όλους τους επισκέπτες του Συστήματος. Αντίθετα κάθε επισκέπτης τους οποίους έχουμε προαναφέρει (Student, Teacher) πρέπει να τρέχει κάτω από μία υπηρεσία Πελάτη (University Client) μέσω της οποίας θα γίνονται όλες οι αιτήσεις του κάθε επισκέπτη και θα εξασφαλίζεται η επικοινωνία του χρήστη με το απομακρυσμένο Πανεπιστήμιο.

Εδώ κρίνεται σκόπιμο να κάνουμε ένα σαφή διαχωρισμό ανάμεσα στις υπηρεσίες Τηλεκπαίδευσης και στις υπηρεσίες Δικτυακής επικοινωνίας.

- Υπηρεσίες Τηλεκπαίδευσης θα εννοούμε κάθε υπηρεσία που έχει καθοριστεί να εξυπηρετεί κάποια ανάγκη της Τηλεκπαίδευσης ενώ
- Υπηρεσίες Επικοινωνίας θα εννοούμε κάθε υπηρεσία που εξυπηρετεί την επικοινωνία Διακομιστή – Πελάτη για την χρήση συγκεκριμένης υπηρεσίας Τηλεκπαίδευσης.

Έτσι το University θα είναι ο παροχέας των υπηρεσιών Τηλεκπαίδευσης τις οποίες θα χρησιμοποιούνε οι χρήστες του Πανεπιστημίου ενώ το University Server θα είναι ο παροχέας των υπηρεσιών επικοινωνίας τις οποίες θα χρησιμοποιούνε οι πελάτες του Συστήματος μέσω των οποίων υλοποιούνται οι υπηρεσίες Τηλεκπαίδευσης.



Εικόνα 7.2 Αναπαράσταση της Δικτυακής Σχεδίασης του Συστήματος Τηλεκπαίδευσης

Εξετάζοντας τις απαιτήσεις του Εικονικού Δικτύου που σχεδιάσαμε (Εικόνα 7.2) βλέπουμε πως τα στοιχεία που απαρτίζουν το Σύστημα μας είναι ένας Υπολογιστής Διακομιστής και ένα σύνολο υπολογιστών Πελατών. Στο κάθε υπολογιστή Πελάτη θα πρέπει να τρέχει και η ανάλογη υπηρεσία Πελάτη. Έτσι ο University Client θα πρέπει όπως είπαμε να υλοποιεί τις διασυνδέσεις Χρήστη – Πανεπιστημίου και Πελάτη – Διακομιστή. Η διαφοροποίηση όμως που θέσαμε στα είδη των χρηστών και των απαιτήσεών τους από το Πανεπιστήμιο και γενικά στις υπηρεσίες Τηλεκπαίδευσης απαιτεί διαφοροποίηση και του Πελάτη (UniversityClient) ανάλογα με το χρήστη που εξυπηρετεί. Ένας Μαθητής χρειάζεται διαφορετικά στοιχεία από το Πανεπιστήμιο από ένα Καθηγητή και ένας Μαθητής χρησιμοποιεί διαφορετικές υπηρεσίες του Πανεπιστημίου από το Καθηγητή.

Ας θέσουμε ορισμένες υπηρεσίες διαθέσιμες από το Εικονικό Πανεπιστήμιο για να δούμε πως αυτές χρησιμοποιούνται από τους διαφοροποιημένους χρήστες. Επειδή η υλοποίηση του Συστήματος μας θα γίνει με την τεχνολογία RMI θα ορίσουμε τις υπηρεσίες αυτές ως μεθόδους (συναρτήσεις) ενός αντικείμένου Διακομιστή τις οποίες οι χρήστες (Πελάτες) θα μπορούν να χρησιμοποιούν απομακρυσμένα. Ο Πελάτης UniversityClient είναι ένα interface της Java στο οποίο δηλώνονται όλες οι διαθέσιμες υπηρεσίες που μπορεί να χρησιμοποιήσει ένας χρήστης. Η δήλωση των συναρτήσεων (υπηρεσιών) και το πώς αυτές λειτουργούν γίνεται από το Διακομιστή UniversityServer.

Διακομιστής UniversityServer

```
public class UniversityServer() extends UnicastRemoteObject
                                    implements UniversityClient {
  private University university = new University();
  public UniversityServer() throws RemoteException {
  }
  public int Login(User user) throws RemoteException {
    return (university.Login(user));
  }
  public Logout(int id) throws RemoteException {
    university.Logout(id);
  }
  public DailyProgram GetDailyProgram(User user) throws RemoteException{
    return (university.GetDailyProgram(user));
  }
  public DailyProgram GetDailyProgram(Student student) throws
                                                    RemoteException {
    return (university.GetDailyProgram(student));
  }
  public DailyProgram GetDailyProgram(Teacher teacher) throws
                                                    RemoteException {
    return (university.GetDailyProgram(teacher));
  }
}
```

Πελάτης UniversityClient

```
public interface UniversityClient extends Remote {
  public int Login(User user) throws RemoteException;
  public void Logout(int id) throws RemoteException;
  public DailyProgram GetDailyProgram(User user) throws RemoteException;
  public DailyProgram GetDailyProgram(Student student) throws
                                                    RemoteException;
  public DailyProgram GetDailyProgram(Teacher teacher) throws
```

```
                                                    RemoteException;
}
```

Ο Διακομιστής UniversityServer διαθέτει δύο υπηρεσίες για τις περιπτώσεις Connect και Disconnect των χρηστών (Login, Logout) και τρεις υπερφορτωμένες συναρτήσεις παροχής του ημερήσιου προγράμματος των χρηστών. Τις συναρτήσεις Login, Logout μπορεί να τις χρησιμοποιήσει οποιοσδήποτε χρήστης για να συνδεθεί στο Σύστημα (η επιστροφή του ακεραίου που γίνεται στο Login είναι μια ταυτότητα ID για τον τρέχον χρήστη), και να αποσυνδεθεί. Όπως είναι φανερό την συνάρτηση GetDailyProgram(User user) μπορούν να τη χρησιμοποιήσουν όλοι οι χρήστες μιας και ο καθένας είναι και ένας User στο Σύστημα μας και μπορεί να εισάγει τα ανάλογα στοιχεία στη συνάρτηση. Το University θα πρέπει να αναγνωρίσει την ιδιότητα του user και να επιστρέψει το σωστό πρόγραμμα. Τις δύο άλλες συναρτήσεις μπορούν να τις χρησιμοποιήσουν μόνο ένας Student και Teacher αντίστοιχα αφού απαιτούνε πιο αυστηρά στοιχεία εισαγωγής. Είναι φανερό πως αν ένας Student προσπαθήσει να χρησιμοποιήσει τη συνάρτηση GetDailyProgram(Teacher teacher) θα υπάρξει πρόβλημα αφού δεν θα μπορεί να δώσει σωστή είσοδο στο πεδίο teacher.

Οι συναρτήσεις αυτές είναι ορατές στους χρήστες και αυτοί μπορούν να τις χρησιμοποιήσουν για την επικοινωνία με το Πανεπιστήμιο. Το Σύστημα Διακομιστή / Πελάτη που δημιουργήσαμε θα δουλέψει καλά. Το μόνο πρόβλημα είναι πως οι υπηρεσίες (συναρτήσεις) είναι συγκεντρωμένες και διαθέσιμες σε όλους. Δηλαδή ακόμα κι αν κάποιος Student δεν μπορέσει ποτέ να χρησιμοποιήσει την συνάρτηση GetDailyProgram(Teacher teacher) αυτή του είναι διαθέσιμη. Αυτό συμβαίνει όπως είδαμε γιατί οι χρήστες σε αυτή την περίπτωση χρησιμοποιούνε την ίδια υπηρεσία Πελάτη UniversityClient.

Οι αλλαγές που πρέπει να γίνουν στον κώδικά μας είναι να προσθέσουμε άλλους δύο Πελάτες UniversityStudentClient και UniversityTeacherClient. Ο Πελάτης UniversityClient θα ανεξαρτητοποιηθεί από κάθε υπηρεσία που αναφέρεται σε συγκεκριμένο χρήστη (Student, Teacher) και θα περιέχει μόνο τις γενικές υπηρεσίες διαθέσιμες σε όλους. Οι δύο καινούργιοι Πελάτες UniversityStudentClient και UniversityTeacherClient θα περιέχουν όλες τις αντίστοιχες υπηρεσίες ενός Student και Teacher αντίστοιχα ενώ θα κληρονομούν τις βασικές υπηρεσίες από τον UniversityClient. Οι δηλώσεις των συναρτήσεων στο Διακομιστή θα παραμείνουν οι

ίδιες ενώ θα πρέπει να οριστούν στην γραμμή των implements οι δύο νέοι Πελάτες τους οποίους εξυπηρετεί ο UniversityServer. Έτσι ο κώδικάς μας έχει ως εξής.

Διακομιστής UniversityServer

```
public class UniversityServer() extends UnicastRemoteObject implements
    UniversityClient, UniversityStudentClient, UniversityTeacherClient {
 private University university = new University();
 public UniversityServer() throws RemoteException {
 }
 public int Login(User user) throws RemoteException {
  return (university.Login(user));
 }
 public Logout(int id) throws RemoteException {
  university.Logout(id);
 }
 public DailyProgram GetDailyProgram(User user) throws RemoteException{
  return (university.GetDailyProgram(user));
 }
 public DailyProgram GetDailyProgram(Student student) throws
                                                    RemoteException {
  return (university.GetDailyProgram(student));
 }
 public DailyProgram GetDailyProgram(Teacher teacher) throws
                                                    RemoteException {
  return (university.GetDailyProgram(teacher));
 }
}
```

Πελάτης UniversityClient

```
public interface UniversityClient extends Remote {
 public int Login(User user) throws RemoteException;
 public void Logout(int id) throws RemoteException;
 public DailyProgram GetDailyProgram(User user) throws RemoteException;
```

```
}
```

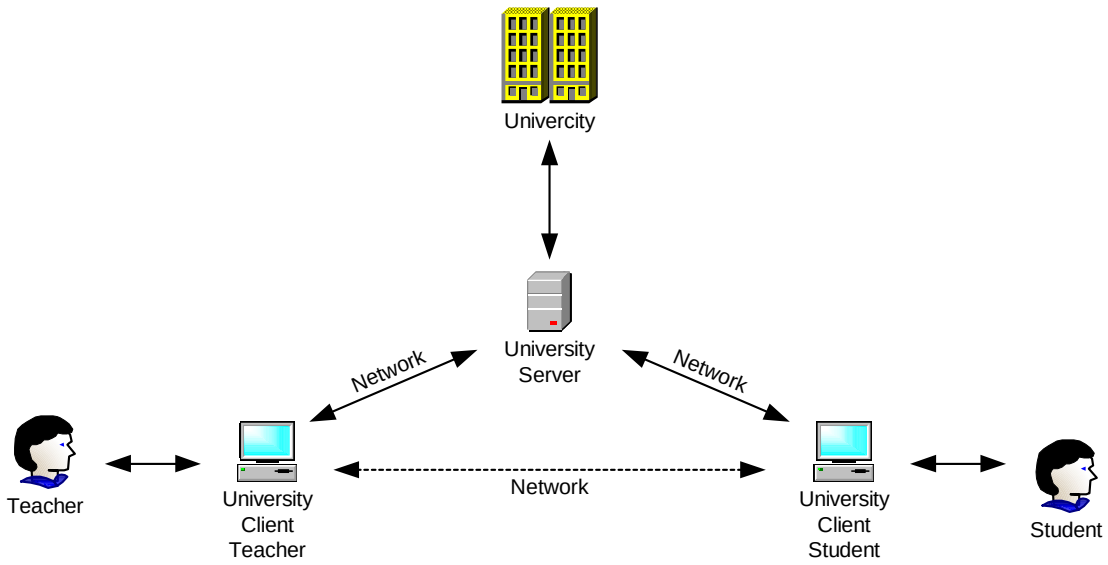
Πελάτης UniversityStudentClient

```
public interface UniversityStudentClient extends UniversityClient {
  public DailyProgram GetDailyProgram(Student student) throws
                                                    RemoteException;
}
```

Πελάτης UniversityTeacherClient

```
public interface UniversityTeacherClient extends UniversityClient {
  public DailyProgram GetDailyProgram(Teacher teacher) throws
                                                    RemoteException;
}
```

Η εικόνα 7.3 μας δείχνει τις δικτυακές απαιτήσεις με τους δύο νέους Πελάτες



Εικόνα 7.3 Αναπαράσταση της Δικτυακής Σχεδίασης του Συστήματος Τηλεκπαίδευσης

7.2 Client μία προσομοίωση Server

Εξετάσαμε παραπάνω το βασικό τρόπο επικοινωνίας Πελατών με ένα απλό Διακομιστή του Συστήματος. Είδαμε τον τρόπο δηλαδή με τον οποίο οι χρήστες του Πανεπιστημίου μπορούν να αντλήσουν Πληροφορίες απομακρυσμένα. Σε ένα σύστημα Τηλεκπαίδευσης όμως δεν είναι μόνο ο Μαθητής ή ο Καθηγητής ο οποίος χρειάζεται πληροφορίες. Ακόμα και το ίδιο το Εικονικό Πανεπιστήμιο πολλές φορές χρειάζεται να αντλεί πληροφορίες, δεδομένα και στοιχεία απομακρυσμένα από τους χρήστες του. Το Σύστημα πρέπει να παρέχει την δυνατότητα αυτή στο Διακομιστή να αντλεί τις απαραίτητες πληροφορίες από τους Πελάτες του και γενικά από τους χρήστες του Πανεπιστημίου ανά πάσα στιγμή.

Ως τώρα έχουμε πει πως αιτήσεις Πελατών εξυπηρετεί μόνο ένας Διακομιστής. Η ανάγκη όμως του ίδιου του Διακομιστή για επικοινωνία με κάποιο Πελάτη του, δημιουργεί την ανάγκη ένας Διακομιστής να μπορεί να στέλνει αιτήσεις σε πελάτες και ταυτόχρονα ένας Πελάτης να μπορεί να εξυπηρετήσει αιτήσεις. Αυτό σημαίνει πως ένας Πελάτης θα πρέπει να είναι ταυτόχρονα και ένας Διακομιστής. Στην ουσία σημαίνει πως ένας Πελάτης απλός πρέπει να εξυπηρετεί τις αιτήσεις του Διακομιστή του και όχι αιτήσεις άλλων Πελατών. Έτσι θα μπορούσαμε να πούμε πως ένας Πελάτης θα είναι η Προσομοίωση ενός Διακομιστή με περιορισμένες δυνατότητες και η υλοποίηση του γίνεται για τις ανάγκες αμφίδρομης επικοινωνίας μεταξύ Διακομιστή – Πελατών, Πανεπιστημίου – Χρηστών.

Ας δούμε το προηγούμενο παράδειγμα Διακομιστή – Πελατών. Θεωρούμε πως ο Διακομιστής θα πρέπει να έχει την δυνατότητα να επικοινωνήσει με τους πελάτες του για να ενημερώσει τους χρήστες ότι το Ημερήσιο Πρόγραμμα του χρήστη έχει αλλάξει. Έτσι θα πρέπει να δημιουργήσουμε μία υπηρεσία στο Πελάτη με την οποία ο Διακομιστής θα επικοινωνεί και θα επεξεργάζεται μία σημαία (flag) στην πλευρά του Πελάτη. Μόλις ο Πελάτης αντιληφθεί ότι έχει αλλάξει η σημαία, τότε θα πρέπει να ξαναζητήσει το καινούριο Ημερήσιο Πρόγραμμα από το Διακομιστή. Ένα καινούριο interface Πελάτη θα πρέπει να δημιουργηθεί έτσι ώστε να εξυπηρετείται ο Διακομιστής. Ονομάζουμε το interface UniversityServerClient. Από την άλλη θα πρέπει να δημιουργήσουμε για το κάθε τύπο Πελάτη και ένα απλό Διακομιστή για την παροχή της συγκεκριμένης υπηρεσίας στο Διακομιστή. Για το συγκεκριμένο παράδειγμα θα χρησιμοποιήσουμε τον πρωτεύον τύπο Πελάτη, το UniversityClient. Οι αλλαγές που πρέπει να γίνουν στο κώδικά είναι.

Διακομιστής UniversityServer

```
public class UniversityServer() extends UnicastRemoteObject implements
    UniversityClient, UniversityStudentClient, UniversityTeacherClient {
 private University university = new University();
 public UniversityServer() throws RemoteException {
 }
 public int Login(User user) throws RemoteException {
  return (university.Login(user));
 }
 public Logout(int id) throws RemoteException {
  university.Logout(id);
 }
 public DailyProgram GetDailyProgram(User user) throws RemoteException{
  return (university.GetDailyProgram(user));
 }
}
```

Πελάτης - Διακομιστής UniversityClientServer

```
public UniversityClientServer extends UnicastRemoteObject
                        implements UniversityServerClient {
 private DailyProgramFlag = false;
 public UniversityClientServer() throws RemoteException {
 }
 public SetDailyProgramChanged(boolean flag) throws RemoteException {
  DailyProgramFlag = flag;
 }
 public boolean GetDailyProgramStatus() {
  return (DailyProgramFlag);
 }
}
```

Πελάτης UniversityClient

```
public interface UniversityClient extends Remote {
```

```
  public int Login(User user) throws RemoteException;
  public void Logout(int id) throws RemoteException;
  public DailyProgram GetDailyProgram(User user) throws RemoteException;
}
```

Πελάτης UniversityServerClient

```
public interface UniverityClient extends Remote {
  public SetDailyProgramChanged(boolean flag) throws RemoteException;
}
```

Όπως φαίνεται από τον κώδικα η επικοινωνία του Διακομιστή με τον Πελάτη γίνεται μέσω της υπηρεσίας SetDailyProgram(boolean flag). Έτσι αν αλλάξει το πρόγραμμα κάποιου χρήστη ο Διακομιστής θα πρέπει να επικοινωνήσει με τον συγκεκριμένο πελάτη και να τρέξει τη συγκεκριμένη συνάρτηση δίνοντας είσοδο 'true' (αλλαγή Προγράμματος). Στη συνέχεια ο Πελάτης θα πρέπει να αντιληφθεί την αλλαγή του προγράμματος ελέγχοντας την σημαία του από τη συνάρτηση GetDailyProgramStatus() και να ξαναζητήσει το καινούριο από το Διακομιστή.

Έως τώρα εξετάσαμε έναν απλό τρόπο υλοποίησης ενός διακομιστή και των Πελατών του με τη χρήση του RMI και το πώς δηλώνονται οι διάφορες υπηρεσίες για την επικοινωνία. Αυτό που δεν είδαμε είναι πώς πραγματοποιείται η απομακρυσμένη κλήση μεθόδου και το πώς το κάθε αντικείμενο επικοινωνεί με ένα άλλο απομακρυσμένο ανάμεσα στο Διαδίκτυο. Θα αναλύσουμε το θέμα αυτό διεξοδικά στο Κεφάλαιο 7.5.

7.3 Ανεξαρτητοποίηση των Υπηρεσιών

Στο προηγούμενο κεφάλαιο είδαμε πώς μπορούν να φτιαχτούν οι υπηρεσίες σε Διακομιστή και Πελάτη για την επικοινωνία μεταξύ τους και εξυπηρέτηση των αναγκών της Τηλεκπαίδευσης. Οι υπηρεσίες αυτές δηλώνονται σε κάποιο interface Πελάτη και υλοποιούνται στο κορμό του Διακομιστή. Με άλλα λόγια όλες οι υπηρεσίες που έχει στη διάθεση του κάποιος χρήστης είναι συγκεντρωμένες στο interface του Πελάτη του είτε αυτός είναι Μαθητής, είτε Καθηγητής, είτε άλλος χρήστης. Ένα Σύστημα Τηλεκπαίδευσης όμως όπως έχουμε προαναφέρει θα πρέπει να υλοποιεί τους δύο τρόπους Τηλεκπαίδευσης, Σύγχρονης και Ασύγχρονης. Είναι

φανερό πως τα δύο είδη αυτά της εκπαίδευσης, χρησιμοποιούνε διαφοροποιημένες υπηρεσίες τόσο όσο στη φύση της Εκπαίδευσης που προσφέρουν όσο και από τον τρόπο που θα πρέπει να τις διαχειρισθεί το Σύστημα. Κρίνεται αναγκαίο λοιπόν να ομαδοποιηθούνε οι διάφορες υπηρεσίες και να ανεξαρτητοποιηθούνε ανάλογα με το είδος και την χρήση τους. Η ομαδοποίηση αυτή έγκειται στο συμπέρασμα πως πρέπει να δημιουργηθεί και ανάλογος αριθμός Διακομιστών με τις ομάδες υπηρεσιών. Δηλαδή ένας Διακομιστής πρέπει να είναι ένα σύνολο Διακομιστών ο καθένας από τους οποίους προσφέρει συγκεκριμένες υπηρεσίες. Από την άλλη πλευρά το ίδιο πρέπει να συμβαίνει και με τον Πελάτη. Ο κάθε Πελάτης πρέπει να είναι ένα σύνολο Πελατών, καθένας από τους οποίους επικοινωνεί με τον αντίστοιχο Διακομιστή και αντλεί τις ανάλογες πληροφορίες.

7.4 Διακομιστής – Πελάτης και Διαχειριστής

Όπως είδαμε παραπάνω ένας Διακομιστής θα πρέπει να αποτελείται από ένα πλήθος Διακομιστών ο καθένας από τους οποίους θα εξυπηρετεί ορισμένες από τις υπηρεσίες του Συστήματος. Εδώ θα εξετάσουμε τις νέες οντότητες που θα πρέπει να δημιουργήσουμε και θα αναλύσουμε το πώς θα πρέπει να αναπροσαρμοστούνε οι υπηρεσίες.

Διακομιστής Server : Τον όρο Server θα τον χρησιμοποιήσουμε για τον κύριο Διακομιστή του Συστήματός μας. Τους διακομιστές που θα βρίσκονται κάτω από τον Server θα τους ονομάσουμε Διαχειριστές (Managers) μίας και η λειτουργία τους θα είναι να διαχειρίζονται ορισμένο αριθμό υπηρεσιών. Ο Διακομιστής λοιπόν θα ασχολείται με την διαχείριση των διαχειριστών και τον έλεγχο της λειτουργίας τους και θα ανεξαρτητοποιηθεί από κάθε υπηρεσία Τηλεκπαίδευσης. Θα μπορεί να κρατάει χρήσιμες πληροφορίες ελέγχου του συστήματος και της κατάστασής του ενώ θα πρέπει να μπορεί να εξασφαλίζει τη σταθερότητά του και την καλή λειτουργία του.

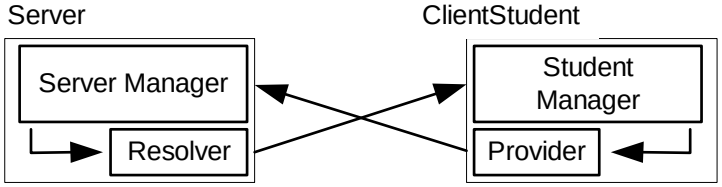
Πελάτης Client : Τον όρο Client θα το χρησιμοποιήσουμε για τον βασικό Πελάτη του Συστήματος. Τους Πελάτες που θα βρίσκονται κάτω από τον Client θα τους ονομάζουμε Διαχειριστές (Managers) μιας και θα διαχειρίζονται ορισμένο αριθμό υπηρεσιών. Ο Client θα ασχολείται με τη διαχείριση των Διαχειριστών του και τον έλεγχο της λειτουργίας τους και θα ανεξαρτητοποιηθεί από κάθε υπηρεσία Τηλεκπαίδευσης. Θα μπορεί να κρατάει χρήσιμες πληροφορίες για την κατάστασή

του ενώ θα πρέπει να εξασφαλίζει την καλή λειτουργία και επικοινωνία με τον Διακομιστή.

Διαχειριστής Manager : Manager θα είναι κάθε Διαχειριστής ορισμένων λειτουργιών και υπηρεσιών του Συστήματος είτε από την πλευρά του Διακομιστή είτε από την πλευρά του Πελάτη. Οι Managers δηλαδή θα είναι υπεύθυνοι για την εξυπηρέτηση των αιτήσεων των Πελατών από την πλευρά του Διακομιστή και σωστή διαχείριση των αιτήσεων και των πληροφοριών που μεταφέρονται απομακρυσμένα από την πλευρά του Πελάτη. Αυτό σημαίνει ότι ένας Manager Πελάτη θα πρέπει να ζητάει πληροφορίες από το Διακομιστή μόνο όταν αυτό κρίνεται απαραίτητο ενώ θα πρέπει να μπορεί να αποθηκεύει πληροφορίες για να μειώνονται οι αιτήσεις. Εδώ πρέπει να πούμε πως η χρήση των Managers μας επιτρέπει να αποδεσμεύσουμε τον χρήστη ο οποίος δεν χρειάζεται να ξέρει πότε ο Πελάτης του κάνει αίτηση στο Διακομιστή για την απόκτηση κάποιων πληροφοριών και πότε ανακτά τις ήδη αποθηκευμένες τοπικά πληροφορίες. Με αυτό τον τρόπο γίνεται καλύτερη διαχείριση του δικτύου το οποίο δεν υπερφορτώνεται και εξασφαλίζεται η καλή επικοινωνία Πελάτη – Διακομιστή.

Προμηθευτής Provider : Το Provider θα είναι το αντικείμενο μέσω του οποίου ένας Manager Πελάτη θα επικοινωνεί με τον απομακρυσμένο Manager του Διακομιστή και θα προμηθεύεται πληροφορίες. Το Provider δηλαδή θα είναι το interface στο οποίο θα είναι δηλωμένες οι υπηρεσίες ενός Manager και το οποίο στην ουσία θα κάνει τις απομακρυσμένες κλήσεις.

Αναλυτής Resolver : Το Resolver θα είναι το αντικείμενο μέσω του οποίου ένας Manager Διακομιστή θα επικοινωνεί με τον απομακρυσμένο Manager ενός Πελάτη και θα προμηθεύεται πληροφορίες. Το Provider δηλαδή θα είναι το interface στο οποίο θα είναι δηλωμένες οι συναρτήσεις μέσω των οποίων ένας Manager μπορεί να επικοινωνήσει με ένα Πελάτη.



Εικόνα 7.4 Αναπαράσταση Επικοινωνίας Server με ClientStudent

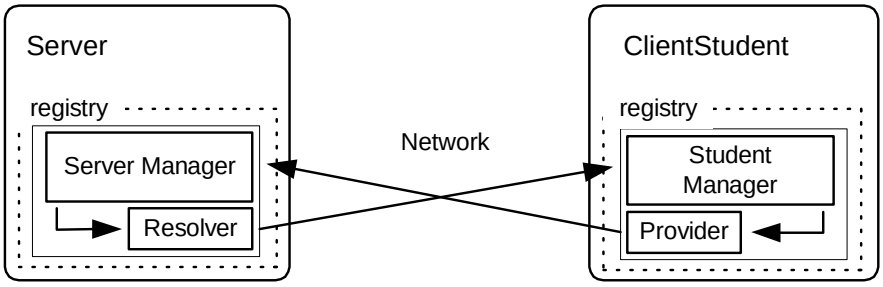
7.5 RMI και Απομακρυσμένη Επικοινωνία

Σε προηγούμενη ενότητα θέσαμε το RMI ως τη βάση επικοινωνίας του Συστήματος μας. Είδαμε με ποιο τρόπο δηλώνονται οι συναρτήσεις προς απομακρυσμένη χρήση και το πώς επιτυγχάνουμε την αμφίδρομη επικοινωνία ανάμεσα σε χρήστες και το Πανεπιστήμιο. Πώς όμως πραγματοποιείται αυτή η επικοινωνία;

Το RMI χρησιμοποιεί μια εικονική περιοχή στη μνήμη του υπολογιστή η οποία ονομάζεται Registry. Η περιοχή αυτή δεσμεύει ένα συγκεκριμένο port του συστήματος έτσι ώστε να μπορούν να γίνονται απομακρυσμένες αναφορές στη συγκεκριμένη μνήμη. Στην εικονική αυτή περιοχή τοποθετούνται όλα τα αντικείμενα τα οποία θέλουμε να κάνουμε διαθέσιμα απομακρυσμένα. Όταν κάποια άλλη απομακρυσμένη πηγή χρειαστεί κάποιο αντικείμενο αρκεί να αναφερθεί στο συγκεκριμένο Socket στο οποίο έχει φορτωθεί το registry της Java και να ζητήσει το αντικείμενο με το όνομα του. Στη συνέχεια μπορεί να χρησιμοποιήσει το αντικείμενο αυτό και τις συναρτήσεις του απομακρυσμένα σαν να βρισκότανε τοπικά στο ίδιο σύστημα.

Στο Σύστημα Τηλεκπαίδευσης στο οποίο σχεδιάζουμε αυτό που χρειάζεται να κάνουμε είναι να τοποθετήσουμε όλες τις υπηρεσίες τις οποίες χρησιμοποιούμε για να επικοινωνούν μεταξύ τους οι χρήστες με το Πανεπιστήμιο σε ένα registry της Java. Έτσι θα μπορούν να γίνονται απομακρυσμένες αναφορές και χρήσεις των υπηρεσιών αυτών και επομένως θα επιτυγχάνεται η επικοινωνία που ζητάμε.

Στο Σύστημά μας οι υπηρεσίες είναι κατηγοριοποιημένες και κατανεμημένες σε αντικείμενα τύπου διαχειριστή (Managers). Συνεπώς κάθε ένας από αυτούς τους Managers θα πρέπει να δηλωθεί και στο συγκεκριμένο registry του Πανεπιστημίου ή του οποιοδήποτε χρήστη. Στην ουσία η απομακρυσμένη επικοινωνία θα γίνεται μεταξύ των Managers οι οποίοι θα είναι διασκορπισμένοι στα κατανεμημένα registry του Συστήματος Τηλεκπαίδευσης. Στη συνέχεια τα δεδομένα θα περνάνε από το Manager στην αντίστοιχη υπηρεσία Πελάτη για να φτάσουνε στην τελική εφαρμογή που θα χειρίζεται ο χρήστης.



Εικόνα 7.5 Αναπαράσταση Επικοινωνίας Server με ClientStudent

7.6 Υλοποίηση Διαχειριστών (Managers)

Παραπάνω είδαμε το βασικό σχεδιασμό των Managers και των Providers του Συστήματος μας αντίστοιχα. Ας δούμε τον τρόπο που αυτοί υλοποιούνται προγραμματιστικά στη Java.

Διαχειριστής ServerManager

```
public class ServerManager extends UnicastRemoteObject implements
                    ClientManager, StudentManager, TeacherManager {
 private ConnectionProperties connectionProperties = new
                                    ConnectionProperties();
 private University university;
 public ServerManager(int port) throws RemoteException {
  university = new University();
  System.setSecurityManager(new RMISecurityManager());
  try {
   java.rmi.registry.LocateRegistry.createRegistry(port);
   Naming.bind("//" + InetAddress.getLocalHost().getHostAddress() + ":" +
                                          port + "/ServerManager", this);
   return (true);
  }
  catch (RemoteException re) {
  }
  catch (Exception e) {
  }
 }
```

```
public ServerProvider CreateServerProvider(String manager, String host, int
                                                    port) throws RemoteException {
  System.setSecurityManager(new RMISecurityManager());
  try {
    return ((ServerProvider)Naming.lookup("//" + host + ":" + port + "/" +
                                                          manager));
  }
  catch (Exception e) {
    return (null);
  }
}
public ConnectionProperties GetProperties() {
  return (connectionProperties);
}
public int LoginStudent(User user) throws RemoteException {
  int LogID = university.LoginStudent(user);
  if (LogID != -1) {
    connectionProperties.SetNewStudentConnected(true);
  }
  return (LogID);
}
public void LogoutStudent(int StudentID) throws RemoteException {
  university.LogoutStudent(StudentID);
  connectionProperties.SetStudentDisconnected(true);
}
public void NewStudentConnected() {
  for (int i = 0; i < university.GetConnectedStudents().Size(); i++) {
    try {
      ServerProvider serverProvider =
        this.CreateServerProvider("StudentManager",
        university.GetConnectedStudents().GetStudent(i).GetHostAddress(),
        2000);
      if (serverProvider != null) {
```

```
            serverProvider.NewStudentConnected();
          }
        }
        catch (RemoteException re) {
        }
      }
    }
    public void StudentDisconnected() {
      for (int i = 0; i < university.GetConnectedStudents().Size(); i++) {
        try {
          ServerProvider serverProvider =
            this.CreateServerProvider("StudentManager",
            university.GetConnectedStudents().GetStudent(i).GetHostAddress(),
            2000);
          serverProvider.StudentDisconnected();
        }
        catch (RemoteException re) {
        }
      }
    }
  }
```

Διαχειρηστής StudentManager

```
public class StudentManager extends UnicastRemoteObject implements
                                        ServerProvider, StudentResolver {
  private ConnectionProperties connectionProperties = new
                                        ConnectionProperties();
  private StudentProvider studentProvider;
  public StudentManager(String host, int port) throws RemoteException {
    try {
      System.setSecurityManager(new RMISecurityManager());
      java.rmi.registry.LocateRegistry.createRegistry(Port));
      Naming.bind("//" + InetAddress.getLocalHost().getHostAddress() + ":" +
```

```
                                                    Port + "/StudentManager", this);
    studentProvider = (StudentProvider)Naming.lookup("//" + host + ":" + port
                                                        + "/ServerManager");
  }
  catch (RemoteException re) {
  }
}
public ConnectionProperties GetProperties() {
 return (connectionProperties);
}
public synchronized int Login(User user) {
 synchronized (this) {
   try {
     return (studentProvider.LoginStudent(user));
   }
   catch (RemoteException re) {
     return (-1);
   }
 }
}
public synchronized void Logout(int id) {
 synchronized (this) {
   try {
     studentProvider.LogoutStudent(id);
   }
   catch (RemoteException re) {
   }
 }
}
public void NewStudentConnected() throws RemoteException {
 connectionProperties.SetNewStudentConnected(true);
}
public void StudentDisconnected() throws RemoteException {
```

```
    connectionProperties.SetStudentDisconnected(true);
  }
}
```

ServerProvider

```
public interface ServerProvider extends Remote {
  public void NewStudentConnected() throws RemoteException;
  public void StudentDisconnected() throws RemoteException;
}
```

StudentProvider

```
public interface StudentProvider extrends ClientProvider {
  public int LoginStudent(User user) throws RemoteException;
  public void LogoutStudent(int id) throws RemoteExcption;
}
```

Παραπάνω φαίνονται οι υλοποιήσεις που πρέπει να γίνουν για την επικοινωνία ενός Πελάτη Μαθητή με το Διακομιστή του Πανεπιστημίου, μέσω της Τεχνολογίας του RMI. Όπως βλέπουμε οι συναρτήσεις επικοινωνίας των απομακρυσμένων αντικειμένων (Managers) δηλώνονται στα αντίστοιχα Providers. Για να επικοινωνήσει ένας Manager με τον αντίστοιχο απομακρυσμένο πρέπει να χρησιμοποιήσει τον ανάλογο Provider που του δίνει αυτή την δυνατότητα.

Για να δηλώσουμε ένα Provider θα πρέπει εξαρχής να γνωρίζουμε το απομακρυσμένο Manager στον οποίο αναφέρεται. Δηλαδή χρειαζόμαστε ένα Socket το οποίο να δείχνει προς το απομακρυσμένο σύστημα στο οποίο είναι δηλωμένο το αντικείμενο που μας ενδιαφέρει. Η αρχικοποίηση ενός απομακρυσμένου αντικείμένου γίνεται μέσω της υπηρεσίας Naming. Στην ουσία μέσω του Naming δηλώνουμε τοπικά ένα απομακρυσμένο αντικείμενο. Το Naming χρησιμοποιεί την lookup για να δηλώσει απομακρυσμένα αντικείμενα και παίρνει εισόδους τον απομακρυσμένο host, το port επικοινωνίας και το όνομα με το οποίο έχει δηλωθεί το αντικείμενο στο registry του απομακρυσμένου συστήματος.

```
studentProvider = (StudentProvider)Naming.lookup("//" + host + ":" + port +
                                                  "/ServerManager");
```

Αν το απομακρυσμένο αντικείμενο δεν έχει πριν δηλωθεί στο registry της απομακρυσμένης συσκευής τότε μία εξαίρεση θα δημιουργηθεί (RemoteException) και η δήλωση δε θα πραγματοποιηθεί. Η ίδια εξαίρεση θα δημιουργηθεί αν το όνομα που δώσαμε στη lookup είναι διαφορετικό από το όνομα που έχει δηλωθεί το απομακρυσμένο αντικείμενο.

Για να δηλώσουμε ένα αντικείμενο σε ένα registry της Java χρησιμοποιούνε την συνάρτηση bind της υπηρεσίας Naming. Πριν από αυτό όμως πρέπει να είμαστε σίγουροι πως η Java έχει δημιουργήσει ένα registry για μας. Η δημιουργία ενός registry γίνεται σε ένα συγκεκριμένο port του συστήματος.

```
java.rmi.registry.LocateRegistry.createRegistry(port);
```

Στην συνέχεια αρκεί να δηλώσουμε το αντικείμενο στο registry το οποίο δημιουργήσαμε. Η συνάρτηση bind παίρνει είσοδο τον host στον οποίο θα δηλωθεί το αντικείμενο, το port επικοινωνίας, το όνομα το οποίο θέλουμε να δώσουμε στο αντικείμενο μέσα στο registry για τις απομακρυσμένες αναφορές και το αντικείμενο που θέλουμε να δηλώσουμε.

```
Naming.bind("//" + InetAddress.getLocalHost().getHostAddress() + ":" + Port
                                                    + "/StudentManager", this);
```

Αν δηλωθούν τα αντικείμενα μέσα στα registry και γίνουν οι απομακρυσμένες δηλώσεις αυτών τότε όπως είπαμε οι κατανεμημένοι Managers μπορούν να επικοινωνούν μεταξύ τους μέσω των υπηρεσιών που των Providers. Ας δούμε όμως ένα κύκλο επικοινωνίας των Managers που υλοποιήσαμε.

Έστω ότι ο Διακομιστής του Πανεπιστημίου ξεκινάει. Τότε δηλώνεται και ένας ServerManager στον Server που τρέχει η υπηρεσία του Διακομιστή. Έστω ότι έχουμε registry που θα χρησιμοποιήσουμε θα είναι στο port 2000. Τότε στην είσοδο δημιουργίας του ServerManager θα πρέπει να δώσουμε 2000.

```
try {
        ServerManager serverManager = new ServerManager(2000);
}
catch (RemoteException re) {
}
```

Με την αρχικοποίηση αυτή του serverManager δημιουργείται αυτόματα ένα registry στο port 2000 και δηλώνεται σ' αυτό ο ίδιος ο serverManager. Από την

άλλη πλευρά ένας μαθητής ξεκινάει την υπηρεσία Πελάτη του. Τότε ένα καινούριο StudentManager δηλώνεται στο port 2000 του συστήματος του μαθητή.

```
try {
        StudentManager studentManager = new StudentManager(Host, 2000);
}
catch (RemoteException re) {
}
```

Με την αρχικοποίηση αυτή του studentManager δημιουργείται αυτόματα ένα registry στο port 2000 στο σύστημα του μαθητή και δηλώνεται σ' αυτό ο ίδιος ο studentManager. Ταυτόχρονα αρχικοποιείται το αντικείμενο studentProvider το οποίο θα αναφέρεται σε ένα αντικείμενο ServerManager στο host ΄Host΄ και στο port 2000.

Τώρα Διακομιστής και Πελάτης Μαθητή τρέχουν στα δύο κατανεμημένα συστήματα και περιμένουν τις απομακρυσμένες αιτήσεις. Έστω ότι ο Μαθητής αποφασίζει να συνδεθεί με το Πανεπιστήμιο. Τότε θα πρέπει να χρησιμοποιήσει τη συνάρτηση Login του studentManger. Ας υποθέσουμε πως τα στοιχεία του χρήστη είναι αποθηκευμένα στο αντικείμενο user.

```
studentManager.Login(user);
```

Με την κλήση της μεθόδου αυτής το studentProvider προσπαθεί να καλέσει την συνάρτηση LoginStudent(user) στον απομακρυσμένο Manager. Αν η κλήση της μεθόδου γίνει χωρίς καμία εξαίρεση (RemoteException) τότε ο Διακομιστής θα προσπαθήσει να συνδέσει τον συγκεκριμένο χρήστη στο University (Πανεπιστήμιο).

```
university.LoginStudent(user);
```

Αν τα στοιχεία του user είναι σωστά και αποδεκτά από το Πανεπιστήμιο τότε θα γίνει επιστροφή ενός αναγνωριστικού ακεραίου από το σύστημα και η σύνδεση θα είναι επιτυχής. Αν όχι τότε θα γίνει επιστροφή του ακεραίου '-1' και ο Πελάτης Μαθητή θα καταλάβει πως η σύνδεση δεν έχει πραγματοποιηθεί επιτυχώς. Στη συνέχεια ο Μαθητής μπορεί να χρησιμοποιεί άλλες υπηρεσίες και να ανακτά δεδομένα για όσο είναι συνδεδεμένος (μέχρι την στιγμή που θα καλέσει την συνάρτηση Logout). Οι υπηρεσίες αυτές θα είναι υλοποιήσιμες και δηλωμένες στα αντικείμενα όπως ακριβώς και η συνάρτηση Login που μόλις είδαμε.

Το connectionProperties που είναι δηλωμένο στους δύο Managers είναι ένα αντικείμενο το οποίο κρατάει καταστάσεις στις συνδέσεις ανάμεσα στους απομακρυσμένους χρήστες. Κάθε φορά που κάποιος καινούριος χρήστης συνδέεται

στο Πανεπιστήμιο οι υπόλοιποι χρήστες θα πρέπει να ενημερωθούν. Ο Διακομιστής θα μπορεί να ελέγξει την κάθε περίπτωση σύνδεσης ή αποσύνδεσης κάποιου χρήστη ελέγχοντας αυτό το αντικείμενο. Στην περίπτωση που ο Διακομιστής καταλάβει πως καινούριος μαθητής συνδέθηκε στο Σύστημα, τότε μπορεί να ενημερώσει τους υπόλοιπους μαθητές χρησιμοποιώντας τη συνάρτηση NewStudentConnected. Η συνάρτηση αυτή όπως φαίνεται από το παραπάνω κώδικα χρησιμοποιεί το ServerProvider για να επικοινωνήσει με τους απομακρυσμένους μαθητές. Με την κλήση της μεθόδου ο Διαχειριστής serverManager δημιουργεί για κάθε χρήστη ένα καινούριο Provider μέσω του οποίου ενημερώνεται το ανάλογο αντικείμενο connectionProperties στην πλευρά του χρήστη. Με αυτό τον τρόπο γίνονται οι ενημερώσεις των χρηστών για τις καταστάσεις του Συστήματος Τηλεκπαίδευσης και γενικά με αυτόν τρόπο επιτυγχάνεται η επικοινωνία Διακομιστή – Πελάτη. Ο Πελάτης τώρα είναι ικανός να γνωρίζει πως ένας καινούριος Μαθητής συνδέθηκε και μπορεί να ενεργήσει αναλόγως. Το πώς γίνεται ο έλεγχος των καταστάσεων σύνδεσης (ConnectionProperties) θα το δούμε στη συνέχεια (Κεφάλαιο 8.7).

7.7 Υλοποίηση Διακομιστή (Server) – Πελάτη (Client)

Όπως έχουμε δει οι Διαχειριστές είναι βασικά συστατικά του Διακομιστή και των Πελατών του μέσω των οποίων επιτυγχάνεται η επικοινωνία ανάμεσα τους. Είδαμε τον τρόπο με τον οποίο υλοποιούνται οι Managers. Μπορούμε τώρα να δούμε τον τρόπο που υλοποιούνται ο Διακομιστής και οι Πελάτες του.

Διακομιστής Server

```
public class Server implements extends Thread implements Runnable {
  private ServerManager serverManager;
  // Άλλοι Managers
  public Server(int port) throws RemoteException {
    try {
      serverManager = new ServerManager(port);
    }
    catch (RemoteException re) {
      throw (re);
    }
```

```
  }
  public ServerManager GetServerManager() {
   return (serverManager);
  }
  public void run() {
  try {
   while (true) {
    if (GetServerManager().GetProperties().NewStudentConnected()) {
     Thread ServerRunner = new Thread() {
      public void run() {
       GetServerManager().NewStudentConnected();
      }
     };
     try {
      ServerRunner.join();
      ServerRunner.start();
     }
     catch (InterruptedException ie) {
     }
      GetServerManager().GetProperties().SetNewStudentConnected(false);
     }
     try {
      sleep(2000);
     }
     catch (InterruptedException ie) {
     }
    }
   }
   catch (Exception e) {
   }
  }
}
```

Πελάτης ClientStudent

```
public class ClientStudent extends Thread implements Runnable {
 private StudentManager studentManager;
 // other Managers
 public ClientStudent(String host, int port) throws RemoteException {
  try {
   studentManager = new StudentManager(host, port);
  }
  catch (RemoteException re) {
   throw (re);
  }
 }
 public StudentManager GetStudentManager() {
  return (studentManager);
 }
 public void run() {
  try {
   while (true) {
    if (GetStudentManager().GetProperties().NewStudentConnected()) {
     Thread ClientRunner = new Thread() {
      public void run() {
       // action
      }
     };
     try {
      ClientRunner.join();
      ClientRunner.start();
     }
     catch (InterruptedException ie) {
     }
     GetStudentManager().GetProperties().SetNewStudentConnected(false);
    }
    try {
```

```
            sleep(2000);
          }
          catch (InterruptedException ie) {
          }
        }
      }
      catch (Exception e) {
      }
    }
  }
```

Παραπάνω βλέπουμε τον τρόπο με τον οποίο υλοποιούνται ένας Διακομιστής (Server) και ένας Πελάτης (ClientStudent). Βασικό χαρακτηριστικό των αντικείμένων αυτών είναι ότι πρέπει να τρέχουν συνέχεια. Αυτό σημαίνει πως τα αντικείμενα αυτά πρέπει να έχουν ελεγχόμενο χρόνο λειτουργίας. Αν σκεφτούμε πως ένα Εικονικό Πανεπιστήμιο λειτουργεί σε Εκπαιδευτικές Περιόδους τότε ο Διακομιστής του πρέπει να είναι ικανός να διεκπεραιώνει αιτήσεις και υπηρεσίες Τηλεκπαίδευσης για όλο το χρονικό διάστημα της χρονικής αυτής περιόδου. Αντίστοιχα ένας Πελάτης θα πρέπει να μπορεί να διατηρεί την επικοινωνία ανάμεσα σε χρήστη και του απομακρυσμένου Συστήματος Τηλεκπαίδευσης για όλο το διάστημα μια συνόδου Τηλεκπαίδευσης. Στην ουσία Διακομιστής και Πελάτης είναι αντικείμενα τα οποία δίνουν ζωή σε ένα Κατανεμημένο Σύστημα απομακρυσμένης επικοινωνίας, μιας και υλοποιούν ελέγχους και διαδικασίες ασταμάτητα μέσα στο χρόνο, ο οποίος θεωρητικά μπορεί να είναι και απεριόριστος.

Την ικανότητα αυτή ο Server και ο ClientStudent να τρέχουν συνέχεια την πετυχαίνουμε χρησιμοποιώντας τη διασύνδεση της Java, Runnable. Το Runnable είναι ένα interface στην Java με το οποίο δημιουργούμε αντικείμενα ανάλογης συμπεριφοράς (runnable).

```
public class Server extends Thread implements Runnable {
        public void run() {
                // Εκτελέσιμος κώδικας
        }
}
```

Για να δηλώσουμε ένα αντικείμενο ως Runnable πρέπει να υλοποιήσουμε τη συνάρτηση run() στο σώμα των αντικειμένων η οποία θα περιέχει τον κώδικα πού θέλουμε να εκτελείται συνέχεια. Μέσα στο σώμα της run() πρέπει να υλοποιήσουμε μία τεχνική η οποία θα κάνει τη συνάρτηση να τρέχει συνέχεια. Αν δεν γίνει αυτό τότε η συνάρτηση θα ξεκινήσει όταν γίνει η κλήση και στη συνέχεια θα σταματήσει της χάνοντας την ιδιότητα Runnable.

```
public void run() {
        while (true) {
                // Εκτελέσιμος κώδικας
        }
}
```

Στο παραπάνω block κώδικα δημιουργούμε ένα ατέρμονο βρόγχο ο οποίος κάνει τη συνάρτηση run να τρέχει συνέχεια. Το μειονέκτημα ενός τέτοιου βρόγχου είναι ότι απορροφάει όλους τους πόρους του συστήματος με το αποτέλεσμα είναι το σύστημα να μην ανταποκρίνεται ή να καταρρεύσει. Τη λύση σ' αυτό το πρόβλημα τη δίνει η συνάρτηση sleep() η οποία παρεμβάλει μία χρονική καθυστέρηση της τάξης milliseconds στην ροή εκτέλεσης ενός νήματος αφήνοντας το σύστημα ελεύθερο να ασχοληθεί με άλλες εργασίες.

Από την πλευρά του Διακομιστή μέσα στη συνάρτηση run() κάνουμε έναν έλεγχο των properties (ConnectionProperties) που αναφέραμε στο προηγούμενο κεφάλαιο. Ο συγκεκριμένος Server ελέγχει μόνο την περίπτωση νέας σύνδεσης ενός μαθητή. Ο έλεγχος αυτός θα εκτελείται συνέχεια για όσο διάστημα θα βρίσκεται σε λειτουργία ο Διακομιστής χάρη της συμπεριφοράς Runnable που δώσαμε στο αντικείμενο. Μόλις ο Διακομιστής αντιληφθεί πως κάποιος καινούριος μαθητής έχει συνδεθεί στο σύστημα ξεκινάει την διαδικασία ενημέρωσης των υπόλοιπων χρηστών. Η ενημέρωση των χρηστών γίνεται μέσω του Διαχειριστή ServerManager και της υπηρεσίας NewStudentConnected την οποία εξετάσαμε σε προηγούμενο κεφάλαιο. Ο Διαχειριστής ServerManager θα προσπαθήσει να επικοινωνήσει με όλους τους μαθητές πράγμα που συνεπάγεται ότι θα υπάρξει μια λογική καθυστέρηση χρόνου στον κύκλο εργασίας του Διακομιστή λόγο της παρεμβολής του Δικτύου στην επικοινωνία. Για να ελαττώσουμε την καθυστέρηση αυτή στο ελάχιστο τοποθετούμε όλη την διαδικασία ενημέρωσης και επικοινωνίας με τους απομακρυσμένους χρήστες σε ένα Νήμα (Thread).

Ένα Thread (νήμα) είναι ένα μέρος ενός προγράμματος που καθορίζεται ώστε να εκτελείται μόνο του, ενώ το υπόλοιπο πρόγραμμα κάνει κάτι άλλο. Αυτό γενικά καλείται πολυεπεξεργασία (Multitasking), επειδή το πρόγραμμα μπορεί να χειριστεί περισσότερες από μία εργασίες ταυτόχρονα. Τα νήματα είναι ιδανικά για οτιδήποτε απαιτεί πολύ χρόνο επεξεργασίας και εκτελείται συνεχώς. Τοποθετώντας τον φόρτο εργασίας μιας διεργασίας σε ένα νήμα, απελευθερώνεται το υπόλοιπο πρόγραμμα, ώστε να χειριστεί άλλα πράγματα. Ταυτόχρονα διευκολύνεται επίσης το περιβάλλον χρόνο εκτέλεσης να χειριστεί το πρόγραμμα, επειδή όλη η εντατική εργασία απομονώνεται στο δικό της νήμα. [1]

Με τη χρήση των νημάτων ο Server μπορεί να διαχειρίζεται πλήθος εργασιών με μηδαμινές χρονικές καθυστερήσεις και να παραμένει σταθερός στη λειτουργία του.

```
Thread TaskRunner = new Thread() {
        public void run() {
                // Εργασία μέσα στο νήμα
        }
};
```

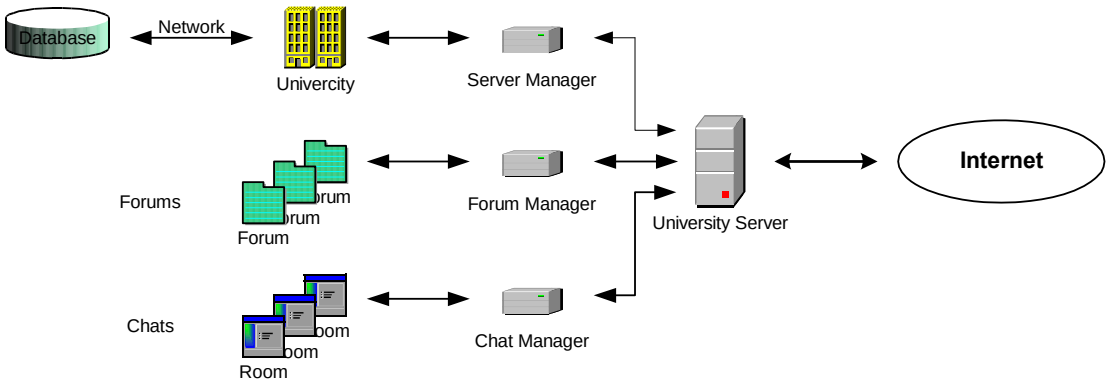
Ο παραπάνω κώδικας είναι η απλή δήλωση ενός καινούριου νήματος. Με τη δήλωση αυτή πετυχαίνουμε μόνο να δημιουργήσουμε μια εργασία και όχι να τη θέσουμε σε λειτουργία. Για να ξεκινήσουμε ένα νήμα χρησιμοποιούμε την συνάρτηση start() του αντικειμένου Thread.

```
try {
        TaskRunner.join();
        TaskRunner.start();
}
catch (InterruptedException ie) {
}
```

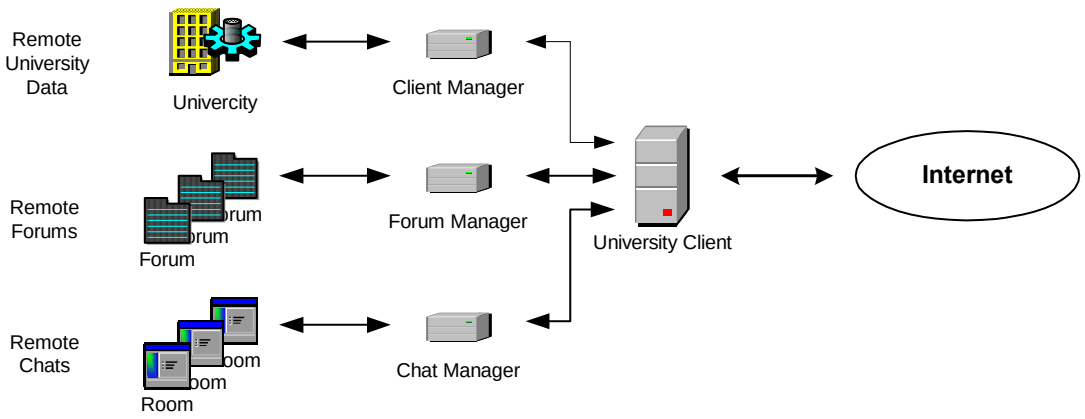
Η συνάρτηση join() ενημερώνει το σύστημα πως ένα καινούριο νήμα έχει δημιουργηθεί και το βάζει σε θέση αναμονής περιμένοντας άλλα νήματα πού ήδη βρίσκονται σε λειτουργία να σταματήσουν, έτσι ώστε το καινούριο νήμα να είναι έτοιμο να ξεκινήσει. Η διαδικασία εκκίνησης ενός νήματος πρέπει να μπει σε ένα block try, catch για την περίπτωση που το καινούριο νήμα αποτύχει να ξεκινήσει. Στην περίπτωση αυτή θα γίνει επιστροφή μιας εξαίρεσης (InterruptedExcepion).

Όπως είδαμε τα αντικείμενα Διακομιστής (Server), Πελάτης (ClientStudent) είναι προγράμματα που δεσμεύουν το Σύστημα για μεγάλο χρονικό διάστημα δεσμεύοντας πόρους συστήματος. Για να απελευθερώσουμε το Σύστημα μπορούμε να εκμεταλλευτούμε τα πλεονεκτήματα που μας δίνουν τα νήματα κληροδοτώντας τις υπηρεσίες που μας προσφέρουν στα ίδια τα αντικείμενα Server, ClientStudent. Με τον τρόπο αυτό Server και ClientStudent φορτώνονται, τρέχουν και διαχειρίζονται από το Σύστημα με μορφή νημάτων.

Στην ουσία Διακομιστής και Πελάτης είναι προγράμματα που τρέχουν στο παρασκήνιο ενός Υπολογιστικού Συστήματος και εξυπηρετούν το Σύστημα Τηλεκπαίδευσης.



Εικόνα 7.6 Αναπαράσταση του Διακομιστή (University Server)



Εικόνα 7.7 Αναπαράσταση του Πελάτη (University Client)

Κεφάλαιο 8ο : Βάση Δεδομένων στο Σύστημα Τηλεκπαίδευσης (Virtual Class)

8.1 Βάση Δεδομένων

Η χρησιμοποίηση κάποιας Βάσης Δεδομένων ήταν απαραίτητο στοιχείο για την ολοκληρωμένη λειτουργία της εφαρμογής. Μερικοί από τους λόγους που την καθιστούν αναντικατάστατη είναι:

- Η ανάγκη καταχώρησης των στοιχείων των χρηστών του Συστήματος και ιδιαίτερα τα στοιχεία των καθηγητών και των μαθητών.
- Η ανάγκη καταχώρησης στοιχείων και πληροφοριών της διαδικασίας της Τηλεκπαίδευσης όπως το Πρόγραμμα Σπουδών, οι Δηλώσεις Μαθημάτων των Μαθητών.
- Η δυνατότητα επεξεργασίας των ήδη καταχωρημένων στοιχείων είτε από τους χρηστές του συστήματος ή τον διαχειριστή.
- Η δυνατότητα να μπορούν να αποθηκεύονται οι βαθμολογίες των Μαθητών και οι προόδους τους στην Τηλεκπαίδευση.

Σε γενικές γραμμές, θα ήταν αδύνατη η δημιουργία μιας τέτοιας εφαρμογής χωρίς να υπάρχει καταχώρηση όλου του όγκου της πληροφορίας σε μια Βάση Δεδομένων.

Στο Σύστημά μας επιλέξαμε ως Βάση Δεδομένων την SQL Server της Microsoft. Η SQL Server είναι μια ολοκληρωμένη εφαρμογή διαχείρισης βάσεων δεδομένων. Βασικό πλεονέκτημα της είναι ότι είναι υλοποιημένη για καθαρά δικτυακές εφαρμογές και μπορεί να εξυπηρετεί αιτήσεις Πελατών άνω των 30, κάτι που δεν μπορούν να κάνουν άλλες Βάσεις (Microsoft Access). Από την άλλη πλευρά η SQL Server προσφέρει ένα σύνολο πιστοποιημένων πολιτικών ασφαλείας για τις βάσεις δεδομένων που υλοποιεί, μέσα από τις οποίες ο καθένας μπορεί να επιλέξει αυτήν που τον ικανοποιεί περισσότερο.

Στην υλοποίηση της Βάσης Δεδομένων του Συστήματος Τηλεκπαίδευσης δε θα κάνουμε ιδιαίτερη αναφορά μιας και η δομή και ο τρόπος λειτουργίας της δεν επηρεάζει άμεσα το Σύστημα, αλλα λειτουργεί ως υποβοήθημα για την ολοκληρωμένη λειτουργία του.

Κεφάλαιο 9^{ο} : Ασφάλεια Συστήματος Τηλεκπαίδευσης (Virtual Class)

9. 1 Ασφάλεια Συστήματος Τηλεκπαίδευσης (Virtual Class)

Το θέμα της ασφάλειας είναι ένα πολύ σημαντικό κομμάτι στο σχεδιασμό και στην υλοποίηση Κατανεμημένων Συστημάτων Πληροφορικής μιας και όλο το σύνολο των πληροφοριών αυτών των Συστημάτων γίνεται διαθέσιμο στους χρήστες του μέσω του Διαδικτύου. Αν και τα Συστήματα αυτά εξυπηρετούνε πάντα εξουσιοδοτημένους χρήστες, δεν μπορεί μέσα στο σχεδιασμό τους να μην συμπεριληφθεί η πιθανότητα εισβολής άλλων χρηστών.

Σε ένα Σύστημα Τηλεκπαίδευσης όπου οι πληροφορίες που διακινούνται στο Διαδίκτυο είναι αρκετά σημαντικές ο καθορισμός κάποιας πολιτικής ασφάλειας στο Σύστημα κρίνεται απαραίτητος.

Σε προηγούμενο κεφάλαιο κάναμε ένα διαχωρισμό των χρηστών του Συστήματος με σκοπό να διερευνήσουμε όλο το εύρος των πιθανών χρηστών του Εικονικού Πανεπιστημίου. Ορίσαμε ένα αντικείμενο Hacker (Εισβολέας) το οποίο θα αντιπροσωπεύει τους χρήστες εκείνους που δεν ενδιαφέρουν το Σύστημα. Η χρήση του αντικειμένου αυτού έγινε για καθαρά σχεδιαστικούς σκοπούς ενώ η υλοποίηση του στο Σύστημα Virtual Class είναι ουσιαστικά τυπική.

Συνεπώς τίθεται το ερώτημα: Υπάρχει Ασφάλεια στο Σύστημα Τηλεκπαίδευσης Virtual Class; Και αν ναι ποια;

Σαφώς και υπάρχει ασφάλεια στο Virtual Class και μάλιστα θα μπορούσαμε να πούμε πως αυτή χωρίζεται σε τρία επίπεδα ασφαλείας.

- Ασφάλεια Βάσης Δεδομένων
- Ασφάλεια Εφαρμογής
- Ασφάλεια Δικτύου

9.2 Ασφάλεια Βάσης Δεδομένων

Το Σύστημα Τηλεκπαίδευσης Virtual Class χρησιμοποιεί τον SQL Server ως Βάση Δεδομένων. Ο SQL Server είναι μία υπηρεσία διαχείρισης βάσεων δεδομένων υλοποιημένη καθαρά για δικτυακές εφαρμογές. Χρησιμοποιεί δικές του πολιτικές ασφαλείας εξυπηρέτησης πελατών μέσα από τις οποίες έχουμε την δυνατότητα να

διαλέξουμε αυτήν που ικανοποιεί το δικό μας Σύστημα. Ως μια ολοκληρωμένη υπηρεσία Διαχείρισης Βάσεων Δεδομένων με την υπογραφή της Microsoft ο SQL Server πιστοποιεί για την εγκυρότητα και την ασφάλεια των αποθηκευμένων πληροφοριών.

Ο SQL Server τρέχει σε δικό του Υπολογιστικό Σύστημα (Server) ανεξάρτητο από το υπόλοιπο Σύστημα Τηλεκπαίδευσης. Μοναδικός πελάτης τον οποίο εξυπηρετεί ο SQL Server είναι ο Διακομιστής (Server) του Συστήματος μέσα από τον οποίο γίνονται οι αιτήσεις της Γραμματείας (Secretariat) για εξυπηρέτηση των χρηστών του Εικονικού Πανεπιστημίου (University). Ο Διακομιστής πιστοποιεί την ταυτότητά του στον SQL Server μέσα από διαδικασία σύνδεσης όπου εισαγάγει δύο συνθηματικά αναγνώρισης (Username, Password). Συνεπώς οι πληροφορίες που εξέρχονται από τη Βάση Δεδομένων είναι προσβάσιμες μόνο από τον ίδιο τον Διακομιστή (Server) του Συστήματος.

9.3 Ασφάλεια Εφαρμογής

Το Σύστημα Τηλεκπαίδευσης που σχεδιάσαμε είναι υλοποιημένο σε παραθυρικό περιβάλλον. Αυτό σημαίνει πως ο κάθε χρήστης για να μπορέσει να συνδεθεί με τον Εικονικό Πανεπιστήμιο και να συμμετάσχει στην Τηλεκπαίδευση χρειάζεται να χρησιμοποιήσει το κατάλληλο λογισμικό. Τα λογισμικά αυτά πρέπει να παρέχονται σε εξουσιοδοτημένους χρήστες έτσι ώστε να επιτυγχάνεται η σωστή χρήση και ασφάλεια του Συστήματος.

Στην περίπτωση που μη εξουσιοδοτημένος χρήστης έχει στη διάθεση του το λογισμικό του Συστήματος που θα του επιτρέψει να συνδεθεί με το Εικονικό Πανεπιστήμιο το ίδιο το Σύστημα εξασφαλίζει την ασφάλειά του. Ο κάθε χρήστης πριν χρησιμοποιήσει οποιαδήποτε πληροφορία του Εικονικού Πανεπιστημίου πρέπει να πρώτα να ταυτοποιήσει τον εαυτό του μέσα από διαδικασία σύνδεσης. Η ταυτοποίηση γίνεται με την εισαγωγή των ανάλογων συνθηματικών (Username, Password) τα οποία είναι μοναδικά για τον κάθε χρήστη. Έτσι στην περίπτωση που κάποιος χρήστης δεν είναι εξουσιοδοτημένος να συνδεθεί στο Σύστημα κάθε προσπάθειά του θα αποτύχει και το Σύστημα θα παραμείνει ασφαλές.

9.4 Ασφάλεια Δικτύου

Ασφάλεια δικτύου είναι φυσικά πολύ δύσκολο να υπάρξει και στις μέρες μας μάλιστα που η Τεχνολογία της Πληροφορικής έχει αναπτυχθεί σε μεγάλο βαθμό μπορεί να θεωρηθεί και ανύπαρκτη.

Τα τελευταία χρόνια έχει αναπτυχθεί μία υπηρεσία μέσω τις οποίας μπορούμε να ορίζουμε ιδιωτικά δίκτυα υπολογιστών μέσα σε άλλα δημόσια. Η υπηρεσία αυτή ονομάζεται Virtual Private Network (VPN) και μιας και το Σύστημά μας είναι υλοποιημένο σε παραθυρικό περιβάλλον εφαρμογής το VPN μοιάζει να είναι ένα καλό εργαλείο ιδιωτικοποίησης του Δικτυακού μας χώρου μέσα στο Διαδίκτυο.

Μέσω του VPN μπορούμε να δηλώσουμε ένα σύνολο απομακρυσμένων Υπολογιστικών Συστημάτων συνδεδεμένων στο Διαδίκτυο σε ένα Εικονικό Δίκτυο με ανάλογη ανάλογα χαρακτηριστικά ενός ιδιωτικού δικτύου. Έτσι οι Υπολογιστές συμπεριφέρονται πλέων σαν να βρίσκονταν τοπικά συνδεδεμένοι σε ένα Δίκτυο στο ίδιο γεωγραφικό χώρο. Με τη δήλωση ενός Υπολογιστικού Συστήματος σε ένα Δίκτυο VPN αποκόπτεται η πρόσβασή του από το δημόσιο δίκτυο και οι πληροφορίες που μπορεί να αντλεί είναι αυτές του ιδιωτικού και μόνο. Στην ουσία το VPN δουλεύει ως ένα δίκτυο αποκομμένο από το δημόσιο μέσα στο οποίο βρίσκεται όπου κανείς δεν έχει πρόσβαση εκτός των Υπολογιστών που είναι δηλωμένοι σε αυτό.

Μέσω του VPN μπορούμε να ορίσουμε ένα εικονικό δικτυακό χώρο στο Internet το οποίο θα παίζει το ρόλο του εικονικού γεωγραφικού χώρου του Εικονικού Πανεπιστημίου. Στο εικονικό αυτό χώρο μπορούν να εισέρχονται μόνο οι εξουσιοδοτημένοι χρήστες του Πανεπιστημίου οι οποίοι συνδέονται με εισαγωγή κατάλληλων συνθηματικών. Έτσι με τον τρόπο αυτό επιτυγχάνεται κάποιας μορφής ασφάλεια αφού οι πληροφορίες Του Συστήματος Τηλεκπαίδευσης γίνονται διαθέσιμες μόνο σε ορισμένους χρήστες από τους πολλούς του Internet.

Κεφάλαιο 10ο: Βελτιώσεις στο Σύστημα Τηλεκπαίδευσης Virtual Class

10.1 Βελτιώσεις στο Σύστημα Τηλεκπαίδευσης Virtual Class

Το Σύστημα Virtual Class είναι όπως είδαμε εργαλείο για παροχή Τηλεκπαίδευσης κάτω από ορισμένες προϋποθέσεις τις οποίες προαναφέραμε. Στην ουσία η Τηλεκπάιδευση μέσω αυτού του Συστήματος γίνεται με τα βασικά χαρακτηριστικά της ασύγχρονης εκπαίδευσης. Γι αυτό το λόγο μπορούμε να πούμε πως το Virtual Class είναι η βάση ενός ολοκληρωμένου Συστήματος Τηλεκπαίδευσης το οποίο μπορεί να υλοποιηθεί με βαθμιαίες βελτιώσεις του πρωταρχικού.

Χαρακτηριστικές βελτιώσεις που μπορούν να γίνουν στο Virtual Class προς την ολοκλήρωσή του αναλύονται στη συνέχεια.

10.2 Χρήση κάθε μέσου Τηλεδιάσκεψης.

Προς το παρόν το Σύστημα Virtual Class χρησιμοποιεί μόνο το Chat το οποίο παίζει το ρόλο της τάξης του Εικονικού Πανεπιστημίου όπου συγκεντρώνονται μαθητές και καθηγητές και συμβιβάζονται με την επικοινωνία αποστολής μόνο απλών μηνυμάτων. Άλλα μέσα όπως ο ήχος ή η εικόνα μπορούν να χρησιμοποιηθούν για να βελτιώσουν τον τρόπο επικοινωνίας των χρηστών και τη διεκπεραίωση της Τηλεκπαίδευσης. Ένα άλλο πολύ χρήσιμο εργαλείο που μπορεί να χρησιμοποιηθεί στην Τηλεκπαίδευση είναι το Whiteboard (Πίνακας). Είναι ένα εργαλείο που παίζει τον ρόλο του πίνακα, πάνω στον οποίο μαθητές και καθηγητές αλληλεπιδρούν γράφοντας και επεξεργάζοντας κείμενα, σημειώσεις και ασκήσεις, ενώ πάνω στο Whiteboard μπορούν να προβληθούν φωτογραφίες και έγγραφα προς διευκόλυνση της Επικοινωνίας. Γενικά η χρήση των μέσων αυτών σε εφαρμογές ονομάζεται Video Conference και αποτελούν εργαλεία Σύγχρονης Εκπαίδευσης αφού επιτρέπουν αμφίδρομη online επικοινωνία μεταξύ χρηστών.

10.3 Χρήση αλυσίδων στα Τηλεμαθήματα.

Οι αλυσίδες είναι ένα θέμα που εξετάσαμε παραπάνω και που ήδη χρησιμοποιούνται στο Σύστημα. Η χρήση τους όπως είδαμε είναι καθαρά πληροφοριακή, για να ξέρει κάποιος μαθητής δηλαδή πως για την κατανόηση της

ύλης κάποιου μαθήματος θα ήταν καλό να παρακολουθήσει τα προαπαιτούμενα μαθήματα πιο πριν. Στην ουσία προς το παρόν οι αλυσίδες δεν δεσμεύουν κάποιο μάθημα αλλά λειτουργούν ως μια επιπλέον πληροφορία του. Επειδή όμως οι αλυσίδες γενικά χρησιμοποιούνται για να δεσμεύσουν κάποιο μάθημα η λειτουργία τους πρέπει να βελτιωθεί. Ένα μάθημα λοιπόν δεν πρέπει να μπορεί να δηλωθεί από κάποιο μαθητή ο οποίος δεν έχει παρακολουθήσει τα προαπαιτούμενά του. Γενικά αυτό μπορεί να επιτευχθεί αν αποθηκεύεται η κατάσταση του κάθε μαθητή. Δηλαδή να μπορεί να ξέρει το Σύστημα ανά πάσα στιγμή ποια μαθήματα έχει παρακολουθήσει κάποιος μαθητής και με τη βαθμό. Έτσι θα μπορεί να γίνει κατάλληλος έλεγχος στη δήλωση μαθημάτων (LessonsDeclaration) των μαθητών και να επιτραπεί η δήλωση ή όχι.

10.4 Υλοποίηση εργαλείων για αξιολόγηση και βαθμολόγηση.

Ως τώρα έχουμε δει πως το Virtual Class δεν ασχολείται με την αξιολόγηση των μαθητευομένων και θέσαμε πως αυτό θα είναι στο χέρι των καθηγητών και θα εξαρτάται πάντα από την πολιτική του Εικονικού Πανεπιστημίου. Έτσι σημαντική βελτίωση στο Σύστημα είναι να δημιουργηθούνε κατάλληλα αντικείμενα – έγγραφα (Documents) τα οποία θα είναι έτοιμα Test τα οποία ο Καθηγητής θα μπορεί να χρησιμοποιεί για να ελέγχει την γνωστική κατάσταση των Μαθητών του. Τα Test θα είναι έτοιμα καλούπια πάνω στα οποία ο κάθε Καθηγητής θα μπορεί να προσαρμόσει το ερωτηματολόγιο που θέλει να θέσει στους μαθητές του. Τα έγγραφα αυτά μπορούν να διαθέτουν κάποιας μορφής χρονοδιακόπτη έτσι ώστε να μπορεί να τεθεί χρονικό περιθώριο στην παράδοση των Test πίσω στους Καθηγητές για αξιολόγηση ενώ οι ερωτήσεις θα μπορούν να είναι της μορφής πολλαπλών επιλογών (Multiple Choice) με δυνατότητα η αξιολόγηση να γίνεται από το Σύστημα. Από την άλλη πλευρά θα πρέπει να δημιουργηθούν αντικείμενα τα οποία θα κρατάνε βαθμολογίες μαθητών ενώ θα πρέπει να γίνουν κατάλληλες τροποποιήσεις στη βάση ώστε να αποθηκεύονται τα νέα δεδομένα.

10.5 Παροχή κάποιας μορφής Πιστοποίησης από το Σύστημα.

Στην περίπτωση που υλοποιηθούνε τρόποι αξιολόγησης των μαθητευομένων τότε στο τέλος της φοίτησης του κάθε μαθητή μπορεί να φανεί η συνολική του πρόοδος και ο τελικός του βαθμός σε κάθε μάθημα και στο σύνολό τους γενικά. Το

Σύστημα θα πρέπει να μπορεί να παρέχει κάποιας μορφής πιστοποίηση για τη φοίτηση του κάθε μαθητευομένου στο Εικονικό Πανεπιστήμιο και για τις γνώσεις που προσκόμισε απ' αυτό. Η πιστοποίηση για παροχή γνώσεων είναι ένα θέμα που εξαρτάται φυσικά από πολλούς παράγοντες όπως για το πόσο σίγουρος μπορεί να είναι κανείς για την ασφάλεια των εξετάσεων μέσα στο Internet, πόσο σίγουρο είναι δηλαδή ότι αυτός που εξετάστηκε ήταν ο εγγεγραμμένος μαθητής και όχι κάποιος άλλος. Σε κάθε περίπτωση πάντως η πιστοποίηση μπορεί να επιτευχθεί με χρήση ανάλογων εγγράφων (Documents) τα οποία θα περιέχουν ψηφιακές υπογραφές και σφραγίδες του Εικονικού Πανεπιστημίου για πιστοποίηση των εγγράφων ενώ η πιστοποίηση των γνώσεων μπορεί να μένει ένα θέμα ανοιχτό.

10.6 Χρήση του Secure Socket Layer (SSL) για ολοκληρωμένη ασφάλεια.

Το SSL είναι ένα πρωτόκολλο ασφαλείας το οποίο προσφέρει κρυπτογράφηση δεδομένων, πιστοποίηση εξυπηρετητή, πελάτη και ακεραιότητα δεδομένων. Είναι συμβατό με dial–in συνδέσεις και χρησιμοποιεί το πρωτόκολλο S-MINE για ασφαλή μετάδοση δεδομένων. Η Java χρησιμοποιεί το SSL σε μία επιπρόσθετη βιβλιοθήκη η οποία ονομάζεται JSSE (Java Secure Socket Extension). Μέσα στη βιβλιοθήκη αυτή υπάρχουν πιστοποιητικά και αλγόριθμοι κρυπτογράφησης τα οποία μπορούν να χρησιμοποιηθούν σε οποιαδήποτε προγραμματιστική εφαρμογή. Μέσω του SSL μπορούμε να προσθέσουμε πολιτικές ασφαλείας στα κλασικά Sockets της Java αλλά και στα Sockets του RMI. Έτσι με το SSL μπορούμε θέσουμε μια πιο ολοκληρωμένη πολιτική ασφαλείας στο Σύστημα Virtual Class οριοθετώντας ότι κάθε απομακρυσμένη σύνδεση του Συστήματος και των χρηστών του θα γίνεται μέσω του Secure Socket Layer.

10.7 Χρήση HTTP-Tunneling για τα Firewalls.

Ένα βασικό μειονέκτημα του Virtual Class είναι η αδυναμία του να επικοινωνήσει με χρήστες που βρίσκονται πίσω από firewalls. Στην περίπτωση ακόμα που ο ίδιος ο Server βρίσκεται πίσω από κάποιο firewall τότε θα είναι αδύνατη η επικοινωνία οποιουδήποτε χρήστη με το Εικονικό Πανεπιστήμιο. Το πρόβλημα αυτό υφίσταται γιατί ο Διακομιστής και οι Πελάτες του Συστήματος χρησιμοποιούνε μη δημόσια port για την επικοινωνία τους (port 2000). Αν και το πρόβλημα μπορεί να λυθεί εύκολα με χρήση γνωστών port τα οποία χρησιμοποιούνται ευρέως στο Internet

και τα οποία επιτρέπονται στη χρήση από τα firewalls (80, 81, 8001, 443) κρίνεται καλύτερη η χρήση του HTTP-Tunneling. Το HTTP-Tunneling επιτρέπει επικοινωνία μεταξύ Διακομιστή και Πελάτη διαμέσου ενός HTTP Server ο οποίος τρέχει στο port 80. Στον HTTP Server πρέπει να υλοποιείται κατάλληλο cgi script (java-rmi.cgi) το οποίο θα ασχολείται με τη διεκπεραίωση της επικοινωνίας των χρηστών και του Εικονικού Πανεπιστημίου. Έτσι σε κάθε περίπτωση που κάποιος Πελάτης αποτύχει να επικοινωνήσει με το Διακομιστή του Εικονικού Πανεπιστημίου λόγο firewall τότε αυτό μπορεί να ξεπεραστεί με το να επικοινωνήσει με τον HTTP Server ο οποίος θα υλοποιεί τις αιτήσεις προς το Διακομιστή για τον Πελάτη.

10.8 Υλοποίηση Site Τηλεκπαίδευσης

Το Σύστημα Τηλεκπαίδευσης Virtual Class υλοποιείται όπως είδαμε μέσω από ένα σύνολο εφαρμογών μία για το κάθε είδους χρήστη (Student, Teacher) και μία για την εφαρμογή του Server του Συστήματος. Ένα ολοκληρωμένο Σύστημα Τηλεκπαίδευσης όμως δεν υφίσταται χωρίς την ύπαρξη κάποιου ανάλογου Site. Έστω ένας μαθητής του Εικονικού Πανεπιστημίου ο οποίος έχει στη διάθεση του κάποιο Υπολογιστικό Σύστημα συνδεδεμένο στο Internet άλλα στο μηχάνημα δεν υπάρχει εγκαταστημένη η εφαρμογή Virtual Class. Φαίνεται αδιανόητο αν ο μαθητής θελήσει να πάρει κάποια πληροφορία από το απομακρυσμένο Πανεπιστήμιο και να μην μπορεί επειδή δεν διαθέτει το πρόγραμμα το συγκεκριμένο μηχάνημα. Έτσι κρίνεται απαραίτητο να υλοποιηθεί κάποιο Site που θα εξυπηρετεί το Εικονικό Πανεπιστήμιο σε ανάλογες περιπτώσεις. Προφανώς μέσω του Site δε θα παρέχεται κάποιας μορφής Τηλεκπαίδευση, αν και αυτό δεν είναι απόλυτο, αλλά θα βασίζεται κυρίως στην παροχή πληροφοριών σε μαθητές, καθηγητές και στους επισκέπτες του. Κάθε χρήστης (Καθηγητής, Μαθητής) θα μπορεί με την εισαγωγή των συνθηματικών του να αντλεί κάθε προσωπική πληροφορία σαν να χρησιμοποιούσε την εφαρμογή του. Στο Site Τηλεκπαίδευσης πρέπει να οριοθετηθούν ανάλογες πολιτικές ασφαλείας με χρήση των SSL για την ασφάλεια των δεδομένων.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Laura Lemay & Rogers Cadenhead, Πλήρες Εγχειρίδιο της Java 2 Platform, Ελληνική έκδοση Γκιούρδα, ISBN: 960-512-175-1

[2] Εμμ. Σκορδαλάκης, Εισαγωγή στη γλώσσα προγραμματισμού Java Εργαστήριο Λογισμικού Ομάδα Τεχνολογίας Λογισμικού ΕΜΠ

[3] David Flanagan, Java in a nutshell, Deluxe Edition και Java Examples in a Nutshell Εκδόσεις O'Reilly ISBN 1-56592-371-5 ή 0596-000-391

[4] Michael Morrison, Jerry Ablan, Teach Yourself More Java in 21 Days, SAMS, ISBN: 1575213478

[5] Andrew Patzer, Professional Java Server Programming, WROX, ISBN: 1861002777

[6] Ablan Morrison, Teach Yoursef more Java in 21 days, Sams Net Shishir Gundavaram, CGI Programming for the World Wide Web, O'reilly

[7] William Horton, Web-based training, Wiley, ISBN 0-471-35614-x

[8] Gilly Salmon, E-moderating: the key to teaching and learning online, Kogan Page, ISBN 0-7494-3110-5

[9] Beverly Abbey, Instructional and cognitive Impacts of Web based education, IGP, ISBN 1-878-28959-4

[10] Peraya, D. (1994). "Distance Education and the WWW." Διαθέσιμο στην διεύθυνση: http://tecfa.unige.ch/edu-ws94/contrib/peraya.fm.html#HDR0

[11] Steiner, V. (1996). "What is Distance Education?" (Far West Laboratory for Educational Research and Development). Διαθέσιμο στην διεύθυνση : http://www.fwl.org/edtech/distance.html

[12] Χ. Σκουρλά, Σχεσιακές Βάσεις Δεδομένων, Αθήνα

ΠΑΡΑΡΤΗΜΑ

Ενδεικτικός Κώδικας

Α. Αντικείμενο Secretariat

```
package learningclass.*;
import java.util.*;
import java.sql.*;
import java.lang.*;
import learningclass.users.*;
import learningclass.documents.*;
import learningclass.lessons.*;
import learningclass.tools.datetime.Day;
import learningclass.tools.datetime.DateTimer;
import learningclass.tools.database.ConnectionPool;
import learningclass.university.managers.LessonNoteManager;
import learningclass.university.properties.SecretariatProperties;

public class Secretariat
{
  private SecretariatProperties secretariatProperties = new SecretariatProperties();
  private boolean ChangeListLessons = false;
  private ConnectionPool Pool = new ConnectionPool();
  private ListTeachers listteachers = new ListTeachers();
  private ListStudents liststudents = new ListStudents();
  private ListAdministrators listadministrators = new ListAdministrators();
  private ListLessons listlessons = new ListLessons();
  private ListLessons lessonTasks = new ListLessons();
  private DailyProgram dailyprogram = new DailyProgram();
  private WeeklyProgram weeklyprogram = new WeeklyProgram();
  private ListWeeklyPrograms studentsPrograms = new ListWeeklyPrograms();
  private ListWeeklyPrograms teachersPrograms = new ListWeeklyPrograms();
```

```
 private LessonNoteManager lessonNoteManager = new LessonNoteManager();

 public Secretariat()
 {
  this.LoadListStudents();
  this.LoadListTeachers();
  this.LoadListAdministrators();
  this.LoadListLessons();
  this.LoadUsersLessons();
  this.LoadDailyProgram();
  dailyprogram.LoadDailyProgram();
  this.LoadWeeklyProgram();
  this.LoadStudentsPrograms();
  this.LoadTeachersPrograms();
 }

 public void LoadPool() throws Exception
 {
  if (Pool.GetDriver() == null)
  {
    Pool.SetDriver("sun.jdbc.odbc.JdbcOdbcDriver");
    Pool.SetURL("jdbc:odbc:Virtual Class");
    Pool.SetPoolSize(5);
    Pool.InitializePool();
  }
 }

 public Connection GetConnection() throws Exception
 {
  Connection Conn = null;
  Conn = Pool.GetConnection();
  return (Conn);
 }
```

```
public void ReleaseConnection(Connection Value) throws Exception
{
 try
 {
  Pool.ReleaseConnection(Value);
 }
 catch (Exception e)
 {
  System.err.println(e.getMessage());
 }
}

public SecretariatProperties GetProperties()
{
 return (secretariatProperties);
}

public void SetChangeListLessons(boolean Value)
{
 ChangeListLessons = Value;
}

public ListTeachers GetListTeachers()
{
 return (listteachers);
}

public ListStudents GetListStudents()
{
 return (liststudents);
}
```

```
public ListAdministrators GetListAdministrators()
{
 return (listadministrators);
}

public ListLessons GetListLessons()
{
 if (this.GetChangeListLessons() == true)
 {
  listlessons = new ListLessons();
  this.LoadListLessons();
  this.SetChangeListLessons(false);
 }
 return (listlessons);
}

public DailyProgram GetDailyProgram()
{
 DateTimer datetimer = new DateTimer();
 if (!dailyprogram.GetDate().equals(datetimer.GetSQLDate()))
 {
  dailyprogram = new DailyProgram();
  this.LoadDailyProgram();
  dailyprogram.LoadDailyProgram();
 }
 return (dailyprogram);
}

public DailyProgram GetDailyProgram(DateTimer Value)
{
 DailyProgram TempDailyProgram = new DailyProgram();
 TempDailyProgram = this.LoadDailyProgram(Value);
 return (TempDailyProgram);
```

```
}

public WeeklyProgram GetWeeklyProgram()
{
 DateTimer datetimer = new DateTimer();
 if (!weeklyprogram.GetDate().equals(datetimer.GetSQLDate()))
 {
  weeklyprogram = new WeeklyProgram();
  this.LoadWeeklyProgram();
 }
 return (weeklyprogram);
}

public WeeklyProgram GetWeeklyProgram(DateTimer Value)
{
 WeeklyProgram TempWeeklyProgram = new WeeklyProgram();
 TempWeeklyProgram = this.LoadWeeklyProgram(Value);
 return (TempWeeklyProgram);
}

public ListWeeklyPrograms GetStudentsWeeklyPrograms()
{
 return (studentsPrograms);
}

public WeeklyProgram GetStudentWeeklyProgram(String user)
{
 return (studentsPrograms.GetWeeklyProgram(user));
}

public ListWeeklyPrograms GetTeachersWeeklyPrograms()
{
 return (teachersPrograms);
```

```
}

public WeeklyProgram GetTeacherWeeklyProgram(String user)
{
 return (teachersPrograms.GetWeeklyProgram(user));
}

public ListLessons GetLessonTasks()
{
 return (lessonTasks);
}

public boolean GetChangeListLessons()
{
 return (ChangeListLessons);
}

public String InsertStudent(VirtualStudent Value) throws Exception
{
 long StudentID;
 String Username = new String(Value.GetUsername());
 String Password = new String(Value.GetPassword());
 String Name = new String(Value.GetName());
 String Lastname = new String(Value.GetLastname());
 String Sex = new String(Value.GetSex());
 String Day = new String(Value.GetDay());
 String Month = new String(Value.GetMonth());
 String Year = new String(Value.GetYear());
 String Address = new String(Value.GetAddress());
 String City = new String(Value.GetCity());
 String Country = new String(Value.GetCountry());
 String Zipcode = new String(Value.GetZipcode());
 String Occupation = new String(Value.GetOccupation());
```

```
String Phone = new String(Value.GetPhone());
String Email = new String(Value.GetEmail());
String SQuestion = new String(Value.GetSQuestion());
String SAnswer = new String(Value.GetSAnswer());
String Identification = new String(Value.GetIdentification());
String Check = new String();
try
{
 Check = this.CheckUser(Username);
}
catch (Exception e)
{
 System.err.println(e.getMessage());
}
if (Check.equals("NotUsed"))
{
 Check = "Submited";
 Connection MyConn = null;
 try
 {
  this.LoadPool();
      MyConn = Pool.GetConnection();
      Statement MyStatement = MyConn.createStatement();
      String SQL = "INSERT INTO Students (Username, Password, Name,
Lastname, Sex, BornDay, BornMonth, BornYear, Address, City, Country, Zipcode,
Occupation, Phone, Email, SQuestion, SAnswer, Identification) ";
      SQL = SQL + "VALUES ('" + Username + "','" + Password + "','" + Name +
"','" + Lastname + "','" + Sex + "','" + Day + "','" + Month + "','" + Year + "','" +
Address + "','" + City + "','" + Country + "','" + Zipcode + "','" + Occupation + "','" +
Phone + "','" + Email + "','" + SQuestion + "','" + SAnswer + "','" + Identification +
"')";
      MyStatement.executeUpdate(SQL);
 }
```

```
  catch (SQLException sqle)
  {
   System.err.println(sqle.getMessage());
  }
  catch (ClassNotFoundException cnfe)
  {
   System.err.println(cnfe.getMessage());
  }
  catch (Exception e)
  {
      System.err.println(e.getMessage());
  }
  finally
  {
   try
      {
    if (!MyConn.equals(null))
       {
         Pool.ReleaseConnection(MyConn);
       }
      }
      catch (Exception e)
      {
       System.err.println(e.getMessage());
      }
  }
  this.LoadListStudents();
  this.secretariatProperties.SetStudentInserted(true);
}
else
{
  Check = "NotSubmited";
}
```

```
  return (Check);
 }

 public String InsertTeacher(VirtualTeacher Value)
 {
  long TeacherID;
  String Username = Value.GetUsername();
  String Password = Value.GetPassword();
  String Name = Value.GetName();
  String Lastname = Value.GetLastname();
  String Sex = Value.GetSex();
  String Day = Value.GetDay();
  String Month = Value.GetMonth();
  String Year = Value.GetYear();
  String Address = Value.GetAddress();
  String City = Value.GetCity();
  String Country = Value.GetCountry();
  String Zipcode = Value.GetZipcode();
  String Occupation = Value.GetOccupation();
  String Phone = Value.GetPhone();
  String Email = Value.GetEmail();
  String SQuestion = Value.GetSQuestion();
  String SAnswer = Value.GetSAnswer();
  String Identification = Value.GetIdentification();
  String Check = new String();
  try
  {
   Check  = this.CheckUser(Username);
  }
  catch (Exception e)
  {
   System.err.println(e.getMessage());
  }
```

```
if (Check == "NotUsed")
{
 Check = "Submited";
 Connection MyConn = null;
 try
 {
  this.LoadPool();
     MyConn = Pool.GetConnection();
  Statement MyStatement = MyConn.createStatement();
     String SQL = "INSERT INTO Teachers (Username, Password, Name,
Lastname, Sex, BornDay, BornMonth, BornYear, Address, City, Country, Zipcode,
Occupation, Phone, Email, SQuestion, SAnswer, Identification) ";
     SQL = SQL + "VALUES ('" + Username + "','" + Password + "','" + Name +
"','" + Lastname + "','" + Sex + "','" + Day + "','" + Month + "','" + Year + "','" +
Address + "','" + City + "','" + Country + "','" + Zipcode + "','" + Occupation + "','" +
Phone + "','" + Email + "','" + SQuestion + "','" + SAnswer + "','" + Identification +
"')";
     MyStatement.executeUpdate(SQL);
 }
 catch (SQLException sqle)
 {
     System.err.println(sqle.getMessage());
 }
 catch (ClassNotFoundException cnfe)
 {
  System.err.println(cnfe.getMessage());
 }
 catch (Exception e)
 {
     System.err.println(e.getMessage());
 }
 finally
 {
```

```
        try
        {
          if ( MyConn != null )
        {
            Pool.ReleaseConnection(MyConn);
        }
        }
        catch (Exception e)
        {
          System.err.println(e.getMessage());
      }
    }
    this.LoadListTeachers();
    this.secretariatProperties.SetTeacherInserted(true);
  }
  else
  {
    Check = "NotSubmited";
  }
  return (Check);
}

public String InsertAdministrator(VirtualAdministrator Value) throws Exception
{
  long AdministratorID;
  String Username = new String(Value.GetUsername());
  String Password = new String(Value.GetPassword());
  String Name = new String(Value.GetName());
  String Lastname = new String(Value.GetLastname());
  String Sex = new String(Value.GetSex());
  String Day = new String(Value.GetDay());
  String Month = new String(Value.GetMonth());
  String Year = new String(Value.GetYear());
```

```
String Address = new String(Value.GetAddress());
String City = new String(Value.GetCity());
String Country = new String(Value.GetCountry());
String Zipcode = new String(Value.GetZipcode());
String Occupation = new String(Value.GetOccupation());
String Phone = new String(Value.GetPhone());
String Email = new String(Value.GetEmail());
String SQuestion = new String(Value.GetSQuestion());
String SAnswer = new String(Value.GetSAnswer());
String Identification = new String(Value.GetIdentification());
String Check = new String();
try
{
 Check = this.CheckUser(Username);
}
catch (Exception e)
{
 System.err.println(e.getMessage());
}
if (Check.equals("NotUsed"))
{
 Check = "Submited";
 Connection MyConn = null;
 try
 {
  this.LoadPool();
      MyConn = Pool.GetConnection();
      Statement MyStatement = MyConn.createStatement();
      String SQL = "INSERT INTO Administrators (Username, Password, Name,
Lastname, Sex, BornDay, BornMonth, BornYear, Address, City, Country, Zipcode,
Occupation, Phone, Email, SQuestion, SAnswer, Identification) ";
      SQL = SQL + "VALUES ('" + Username + "','" + Password + "','" + Name +
"','" + Lastname + "','" + Sex + "','" + Day + "','" + Month + "','" + Year + "','" +
```

```
Address + "','" + City + "','" + Country + "','" + Zipcode + "','" + Occupation + "','" +
Phone + "','" + Email + "','" + SQuestion + "','" + SAnswer + "','" + Identification +
"')";
        MyStatement.executeUpdate(SQL);
    }
    catch (SQLException sqle)
    {
     System.err.println(sqle.getMessage());
    }
    catch (ClassNotFoundException cnfe)
    {
     System.err.println(cnfe.getMessage());
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
    finally
    {
     try
        {
       if (!MyConn.equals(null))
          {
            Pool.ReleaseConnection(MyConn);
          }
        }
        catch (Exception e)
        {
          System.err.println(e.getMessage());
        }
    }
    this.LoadListAdministrators();
    this.secretariatProperties.SetAdministratorInserted(true);
```

```
  }
  else
  {
    Check = "NotSubmited";
  }
  return (Check);
}

public String UpdateStudent(VirtualStudent Value) throws Exception
{
  long StudentID;
  String Username = new String(Value.GetUsername());
  String Password = new String(Value.GetPassword());
  String Name = new String(Value.GetName());
  String Lastname = new String(Value.GetLastname());
  String Sex = new String(Value.GetSex());
  String Day = new String(Value.GetDay());
  String Month = new String(Value.GetMonth());
  String Year = new String(Value.GetYear());
  String Address = new String(Value.GetAddress());
  String City = new String(Value.GetCity());
  String Country = new String(Value.GetCountry());
  String Zipcode = new String(Value.GetZipcode());
  String Occupation = new String(Value.GetOccupation());
  String Phone = new String(Value.GetPhone());
  String Email = new String(Value.GetEmail());
  String SQuestion = new String(Value.GetSQuestion());
  String SAnswer = new String(Value.GetSAnswer());
  String Identification = new String(Value.GetIdentification());
  String Check = new String();
  try
  {
    Check = this.CheckUser(Username);
```

```
    }
    catch (Exception e)
    {
     System.err.println(e.getMessage());
    }
    if (Check.equals("Used"))
    {
     Check = "Updated";
     Connection MyConn = null;
     try
     {
         this.LoadPool();
       MyConn = Pool.GetConnection();
         Statement MyStatement = MyConn.createStatement();
         String SQL = "UPDATE Students SET Password = '" + Password + "', Name
= '" + Name + "', Lastname = '" + Lastname + "', Sex = '" + Sex + "', BornDay = '" +
Day + "', BornMonth = '" + Month + "', BornYear = '" + Year + "', Address = '" +
Address + "', City = '" + City + "', Country = '" + Country + "', Zipcode = '" + Zipcode
+ "', Occupation = '" + Occupation + "', Phone = '" + Phone + "', Email = '" + Email +
"', SQuestion = '" + SQuestion + "', SAnswer = '" + SAnswer + "' ";
       SQL = SQL + "WHERE Username = '" + Username + "'";
         MyStatement.executeUpdate(SQL);
     }
     catch (SQLException sqle)
     {
         System.err.println(sqle.getMessage());
     }
     catch (ClassNotFoundException cnfe)
     {
         System.err.println(cnfe.getMessage());
     }
     catch (Exception e)
     {
```

```
      System.err.println(e.getMessage());
    }
    finally
    {
        try
        {
      if (!MyConn.equals(null))
         {
           Pool.ReleaseConnection(MyConn);
         }
     }
        catch (Exception e)
        {
         System.err.println(e.getMessage());
        }
    }
    this.LoadListStudents();
    this.secretariatProperties.SetStudentUpdated(true);
  }
  else
  {
    Check = "NotUpdated";
  }
  return (Check);
}

public String UpdateTeacher(VirtualTeacher Value) throws Exception
{
  long TeacherID;
  String Username = new String(Value.GetUsername());
  String Password = new String(Value.GetPassword());
  String Name = new String(Value.GetName());
  String Lastname = new String(Value.GetLastname());
```

```
String Sex = new String(Value.GetSex());
String Day = new String(Value.GetDay());
String Month = new String(Value.GetMonth());
String Year = new String(Value.GetYear());
String Address = new String(Value.GetAddress());
String City = new String(Value.GetCity());
String Country = new String(Value.GetCountry());
String Zipcode = new String(Value.GetZipcode());
String Occupation = new String(Value.GetOccupation());
String Phone = new String(Value.GetPhone());
String Email = new String(Value.GetEmail());
String SQuestion = new String(Value.GetSQuestion());
String SAnswer = new String(Value.GetSAnswer());
String Identification = new String(Value.GetIdentification());
String Check = new String();
try
{
  Check = this.CheckUser(Username);
}
catch (Exception e)
{
  System.err.println(e.getMessage());
}
if (Check.equals("Used"))
{
  Check = "Updated";
  Connection MyConn = null;
  try
  {
      this.LoadPool();
    MyConn = Pool.GetConnection();
    Statement MyStatement = MyConn.createStatement();
```

```
String SQL = "UPDATE Teachers SET Password = '" + Password + "', Name =
'" + Name + "', Lastname = '" + Lastname + "', Sex = '" + Sex + "', BornDay = '" +
Day + "', BornMonth = '" + Month + "', BornYear = '" + Year + "', Address = '" +
Address + "', City = '" + City + "', Country = '" + Country + "', Zipcode = '" + Zipcode
+ "', Occupation = '" + Occupation + "', Phone = '" + Phone + "', Email = '" + Email +
"', SQuestion = '" + SQuestion + "', SAnswer = '" + SAnswer + "' ";
        SQL = SQL + "WHERE Username = '" + Username + "'";
     MyStatement.executeUpdate(SQL);
    }
    catch (SQLException sqle)
    {
     System.err.println(sqle.getMessage());
    }
    catch (ClassNotFoundException cnfe)
    {
     System.err.println(cnfe.getMessage());
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
    finally
    {
        try
        {
         if (!MyConn.equals(null))
       {
            Pool.ReleaseConnection(MyConn);
       }
     }
        catch (Exception e)
        {
         System.err.println(e.getMessage());
```

```
        }
    }
    this.LoadListTeachers();
    this.secretariatProperties.SetTeacherUpdated(true);
  }
  else
  {
    Check = "NotUpdated";
  }
  return (Check);
}

public String UpdatePrincipal(Principal Value) throws Exception
{
  long TeacherID;
  String Username = new String(Value.GetUsername());
  String Password = new String(Value.GetPassword());
  String Name = new String(Value.GetName());
  String Lastname = new String(Value.GetLastname());
  String Sex = new String(Value.GetSex());
  String Day = new String(Value.GetDay());
  String Month = new String(Value.GetMonth());
  String Year = new String(Value.GetYear());
  String Address = new String(Value.GetAddress());
  String City = new String(Value.GetCity());
  String Country = new String(Value.GetCountry());
  String Zipcode = new String(Value.GetZipcode());
  String Occupation = new String(Value.GetOccupation());
  String Phone = new String(Value.GetPhone());
  String Email = new String(Value.GetEmail());
  String SQuestion = new String(Value.GetSQuestion());
  String SAnswer = new String(Value.GetSAnswer());
  String Identification = new String(Value.GetIdentification());
```

```
String Check = new String();
try
{
 Check = this.CheckUser(Username);
}
catch (Exception e)
{
 System.err.println(e.getMessage());
}
if (Check.equals("Used"))
{
 Check = "Updated";
 Connection MyConn = null;
 try
 {
     this.LoadPool();
  MyConn = Pool.GetConnection();
  Statement MyStatement = MyConn.createStatement();
  String SQL = "UPDATE Teachers SET Password = '" + Password + "', Name =
'" + Name + "', Lastname = '" + Lastname + "', Sex = '" + Sex + "', BornDay = '" +
Day + "', BornMonth = '" + Month + "', BornYear = '" + Year + "', Address = '" +
Address + "', City = '" + City + "', Country = '" + Country + "', Zipcode = '" + Zipcode
+ "', Occupation = '" + Occupation + "', Phone = '" + Phone + "', Email = '" + Email +
"', SQuestion = '" + SQuestion + "', SAnswer = '" + SAnswer + "' ";
     SQL = SQL + "WHERE Username = '" + Username + "'";
     MyStatement.executeUpdate(SQL);
 }
 catch (SQLException sqle)
 {
     System.err.println(sqle.getMessage());
 }
 catch (ClassNotFoundException cnfe)
 {
```

```
            System.err.println(cnfe.getMessage());
        }
        catch (Exception e)
        {
            System.err.println(e.getMessage());
        }
        finally
        {
            try
            {
           if (!MyConn.equals(null))
           {
               Pool.ReleaseConnection(MyConn);
             }
            }
            catch (Exception e)
            {
             System.err.println(e.getMessage());
       }
      }
      this.LoadListTeachers();
      this.secretariatProperties.SetTeacherUpdated(true);
    }
    else
    {
      Check = "NotUpdated";
    }
    return (Check);
  }

  public String UpdateAdministrator(VirtualAdministrator Value) throws Exception
  {
    long AdministratorID;
```

```
String Username = new String(Value.GetUsername());
String Password = new String(Value.GetPassword());
String Name = new String(Value.GetName());
String Lastname = new String(Value.GetLastname());
String Sex = new String(Value.GetSex());
String Day = new String(Value.GetDay());
String Month = new String(Value.GetMonth());
String Year = new String(Value.GetYear());
String Address = new String(Value.GetAddress());
String City = new String(Value.GetCity());
String Country = new String(Value.GetCountry());
String Zipcode = new String(Value.GetZipcode());
String Occupation = new String(Value.GetOccupation());
String Phone = new String(Value.GetPhone());
String Email = new String(Value.GetEmail());
String SQuestion = new String(Value.GetSQuestion());
String SAnswer = new String(Value.GetSAnswer());
String Identification = new String(Value.GetIdentification());
String Check = new String();
try
{
  Check = this.CheckUser(Username);
}
catch (Exception e)
{
  System.err.println(e.getMessage());
}
if (Check.equals("Used"))
{
  Check = "Updated";
  Connection MyConn = null;
  try
  {
```

```
            this.LoadPool();
        MyConn = Pool.GetConnection();
            Statement MyStatement = MyConn.createStatement();
            String SQL = "UPDATE Administrators SET Password = '" + Password + "',
Name = '" + Name + "', Lastname = '" + Lastname + "', Sex = '" + Sex + "', BornDay
= '" + Day + "', BornMonth = '" + Month + "', BornYear = '" + Year + "', Address = '"
+ Address + "', City = '" + City + "', Country = '" + Country + "', Zipcode = '" +
Zipcode + "', Occupation = '" + Occupation + "', Phone = '" + Phone + "', Email = '" +
Email + "', SQuestion = '" + SQuestion + "', SAnswer = '" + SAnswer + "' ";
        SQL = SQL + "WHERE Username = '" + Username + "'";
            MyStatement.executeUpdate(SQL);
        }
        catch (SQLException sqle)
        {
            System.err.println(sqle.getMessage());
        }
        catch (ClassNotFoundException cnfe)
        {
            System.err.println(cnfe.getMessage());
        }
        catch (Exception e)
        {
         System.err.println(e.getMessage());
        }
        finally
        {
            try
            {
          if (!MyConn.equals(null))
              {
                Pool.ReleaseConnection(MyConn);
              }
         }
```

```
        catch (Exception e)
        {
          System.err.println(e.getMessage());
        }
     }
     this.LoadListAdministrators();
     this.secretariatProperties.SetAdministratorUpdated(true);
   }
   else
   {
     Check = "NotUpdated";
   }
   return (Check);
 }

  public VirtualStudent StudentQuestionAnswer(String SQuestion, String SAnswer)
throws Exception
 {
   VirtualStudent TempStudent = null;
   String Check = new String();
   if ((SQuestion.equals(null)) || (SQuestion.equals("")) ||
      (SAnswer.equals(null)) || (SAnswer.equals("")))
        {
       Check = "Empty";
        }
        else
        {
          Connection MyConn = null;
          try
          {
            this.LoadPool();
            MyConn = Pool.GetConnection();
         Statement MyStatement = MyConn.createStatement();
```

```
        String SQL = "SELECT * FROM Students WHERE SQuestion = '" +
SQuestion + "' AND SAnswer = '" + SAnswer + "'";
        ResultSet RS = MyStatement.executeQuery(SQL);
      if (RS.next())
        {
         Check = "Find";
       TempStudent = new VirtualStudent();
       TempStudent.SetStudentID(RS.getLong("StudentID"));
       TempStudent.SetUsername(RS.getString("Username"));
       TempStudent.SetPassword(RS.getString("Password"));
       TempStudent.SetName(RS.getString("Name"));
       TempStudent.SetLastname(RS.getString("Lastname"));
       TempStudent.SetSex(RS.getString("Sex"));
       TempStudent.SetDay(RS.getString("BornDay"));
       TempStudent.SetMonth(RS.getString("BornMonth"));
       TempStudent.SetYear(RS.getString("BornYear"));
       TempStudent.SetAddress(RS.getString("Address"));
       TempStudent.SetCity(RS.getString("City"));
       TempStudent.SetCountry(RS.getString("Country"));
       TempStudent.SetZipcode(RS.getString("Zipcode"));
       TempStudent.SetOccupation(RS.getString("Occupation"));
       TempStudent.SetPhone(RS.getString("Phone"));
       TempStudent.SetEmail(RS.getString("Email"));
       TempStudent.SetSQuestion(RS.getString("SQuestion"));
       TempStudent.SetSAnswer(RS.getString("SAnswer"));
      }
        else
        {
         Check = "NotFind";
      }
      RS.close();
      }
      catch (SQLException sqle)
```

```
        {
          System.err.println(sqle.getMessage());
        }
        catch (ClassNotFoundException cnfe)
        {
          System.err.println(cnfe.getMessage());
        }
        catch (Exception e)
        {
          System.err.println(e.getMessage());
        }
      finally
        {
          try
          {
        if (!MyConn.equals(null))
            {
              Pool.ReleaseConnection(MyConn);
        }
          }
          catch (Exception e)
          {
            System.err.println(e.getMessage());
          }
      }
    }
    return (TempStudent);
  }

  public VirtualTeacher TeacherQuestionAnswer(String SQuestion, String SAnswer)
throws Exception
  {
    VirtualTeacher TempTeacher = null;
```

```
String Check = new String();
if ((SQuestion.equals(null)) || (SQuestion.equals("")) ||
      (SAnswer.equals(null)) || (SAnswer.equals("")))
      {
       Check = "Empty";
      }
      else
      {
       Connection MyConn = null;
       try
       {
         this.LoadPool();
      MyConn = Pool.GetConnection();
      Statement MyStatement = MyConn.createStatement();
      String SQL = "SELECT * FROM Teachers WHERE SQuestion = '" +
SQuestion + "' AND SAnswer = '" + SAnswer + "'";
      ResultSet RS = MyStatement.executeQuery(SQL);
      if (RS.next())
      {
       Check = "Find";
            TempTeacher = new VirtualTeacher();
            TempTeacher.SetTeacherID(RS.getLong("TeacherID"));
            TempTeacher.SetUsername(RS.getString("Username"));
            TempTeacher.SetPassword(RS.getString("Password"));
            TempTeacher.SetName(RS.getString("Name"));
            TempTeacher.SetLastname(RS.getString("Lastname"));
            TempTeacher.SetSex(RS.getString("Sex"));
            TempTeacher.SetDay(RS.getString("BornDay"));
        TempTeacher.SetMonth(RS.getString("BornMonth"));
            TempTeacher.SetYear(RS.getString("BornYear"));
            TempTeacher.SetAddress(RS.getString("Address"));
            TempTeacher.SetCity(RS.getString("City"));
            TempTeacher.SetCountry(RS.getString("Country"));
```

```
            TempTeacher.SetZipcode(RS.getString("Zipcode"));
            TempTeacher.SetOccupation(RS.getString("Occupation"));
            TempTeacher.SetPhone(RS.getString("Phone"));
            TempTeacher.SetEmail(RS.getString("Email"));
            TempTeacher.SetSQuestion(RS.getString("SQuestion"));
            TempTeacher.SetSAnswer(RS.getString("SAnswer"));
  }
  else
  {
   Check = "NotFind";
  }
  RS.close();
}
   catch (SQLException sqle)
   {
    System.err.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
   {
    System.err.println(cnfe.getMessage());
}
   catch (Exception e)
   {
    System.err.println(e.getMessage());
}
   finally
   {
    try
  {
          if (!MyConn.equals(null))
          {
           Pool.ReleaseConnection(MyConn);
          }
```

```
            }
            catch (Exception e)
            {
                    System.err.println(e.getMessage());
            }
          }
        }
        return (TempTeacher);
          }

   public VirtualAdministrator AdministratorQuestionAnswer(String SQuestion,
String SAnswer) throws Exception
   {
    VirtualAdministrator TempAdministrator = null;
    String Check = new String();
    if ((SQuestion.equals(null)) || (SQuestion.equals("")) ||
          (SAnswer.equals(null)) || (SAnswer.equals("")))
          {
           Check = "Empty";
          }
          else
          {
           Connection MyConn = null;
           try
           {
             this.LoadPool();
          MyConn = Pool.GetConnection();
          Statement MyStatement = MyConn.createStatement();
          String SQL = "SELECT * FROM Administrators WHERE SQuestion = '" +
SQuestion + "' AND SAnswer = '" + SAnswer + "'";
          ResultSet RS = MyStatement.executeQuery(SQL);
          if (RS.next())
          {
```

```
            Check = "Find";
            TempAdministrator = new VirtualAdministrator();

TempAdministrator.SetAdministratorID(RS.getLong("AdministratorID"));
            TempAdministrator.SetUsername(RS.getString("Username"));
            TempAdministrator.SetPassword(RS.getString("Password"));
            TempAdministrator.SetName(RS.getString("Name"));
            TempAdministrator.SetLastname(RS.getString("Lastname"));
            TempAdministrator.SetSex(RS.getString("Sex"));
            TempAdministrator.SetDay(RS.getString("BornDay"));
            TempAdministrator.SetMonth(RS.getString("BornMonth"));
            TempAdministrator.SetYear(RS.getString("BornYear"));
            TempAdministrator.SetAddress(RS.getString("Address"));
            TempAdministrator.SetCity(RS.getString("City"));
            TempAdministrator.SetCountry(RS.getString("Country"));
            TempAdministrator.SetZipcode(RS.getString("Zipcode"));
            TempAdministrator.SetOccupation(RS.getString("Occupation"));
            TempAdministrator.SetPhone(RS.getString("Phone"));
            TempAdministrator.SetEmail(RS.getString("Email"));
            TempAdministrator.SetSQuestion(RS.getString("SQuestion"));
            TempAdministrator.SetSAnswer(RS.getString("SAnswer"));
  }
  else
  {
            Check = "NotFind";
  }
  RS.close();
}
catch (SQLException sqle)
    {
      System.err.println(sqle.getMessage());
}
    catch (ClassNotFoundException cnfe)
```

```
            {
             System.err.println(cnfe.getMessage());
        }
            catch (Exception e)
            {
             System.err.println(e.getMessage());
        }
            finally
            {
             try
            {
             if (!MyConn.equals(null))
                    {
                     Pool.ReleaseConnection(MyConn);
             }
            }
            catch (Exception e)
            {
             System.err.println(e.getMessage());
            }
        }
            }
      return (TempAdministrator);
        }

public void InsertStudiesProgram(StudiesProgram Value)
{
  int sixmonths;
  long StudiesProgramID;
  SixMonth sixmonth = new SixMonth();
  Connection MyConn = null;
  try
  {
```

```
            this.LoadPool();
            MyConn = Pool.GetConnection();
            Statement MyStatement = MyConn.createStatement();
            String SQL = "INSERT INTO StudiesProgram (Name,
Maxsixmonthsnumber) ";
        SQL = SQL + "VALUES ('" + Value.GetName() + "', " +
10/*Value.GetMaxSixMonths()*/ + ")";
            MyStatement.executeUpdate(SQL);
            SQL = "SELECT * FROM StudiesProgram WHERE Name = '" +
Value.GetName() + "'";
            ResultSet RS = MyStatement.executeQuery(SQL);
            StudiesProgramID = RS.getLong("StudiesProgramID");
            sixmonths = Value.Size();
            for (int i = 0; i < sixmonths; i++)
            {
              sixmonth = Value.GetSixMonth(i);
              sixmonth.SetStudiesProgramID(StudiesProgramID);
              this.SetStudiesProgramSixMonth(sixmonth);
            }
       RS.close();
     }
     catch (SQLException sqle)
     {
          System.err.println(sqle.getMessage());
     }
     catch (ClassNotFoundException cnfe)
     {
       System.err.println(cnfe.getMessage());
     }
     catch (Exception e)
     {
       System.err.println(e.getMessage());
     }
```

```
    finally
    {
        try
        {
          if ( MyConn != null )
        {
            Pool.ReleaseConnection(MyConn);
          }
        }
        catch (Exception e)
        {
          System.err.println(e.getMessage());
        }
    }
  }

  public void ApplyLessonsDeclaration(LessonsDeclaration declaration) throws
NotValidLessonsDeclarationException, LessonsDeclarationPeriodPassedException
  {
   try
   {
    CheckDeclaration(declaration);
    if (LessonDeclaration.IfPeriodPassed())
   {
    throw (new
LessonsDeclarationPeriodPassedException("LessonsDeclarationPeriodPassedExcepti
on"));
   }
    InsertLessonsDeclaration(declaration);
   }
   catch (NotValidLessonsDeclarationException nvlde)
   {
    throw (nvlde);
```

```
  }
 }

 public LessonsDeclaration GetLessonsDeclaration(String user)
 {
  return (LessonDeclarationManager.GetLessonsDeclaration(user));
 }

 public LessonsDeclaration GetLessonsDeclaration(String user, String period)
 {
  return (LessonDeclarationManager.GetLessonsDeclaration(user, period));
 }

 private synchronized void InsertLessonsDeclaration(LessonsDeclaration declaration)
 {
  for (int i = 0; i < declaration.GetStudiesProgram().Size(); i++)
  {
   for (int j = 0; j < declaration.GetStudiesProgram().GetSixMonth(i).Size(); j++)
   {
    if (declaration.GetStudiesProgram().GetSixMonth(i).GetLesson(j).IsDeclared())
    {
     RemoveLessonStudent(declaration.GetStudent(),
declaration.GetStudiesProgram().GetSixMonth(i).GetLesson(j));
     AddLessonStudent(declaration.GetStudent(),
declaration.GetStudiesProgram().GetSixMonth(i).GetLesson(j));
    }
    else
    {
     RemoveLessonStudent(declaration.GetStudent(),
declaration.GetStudiesProgram().GetSixMonth(i).GetLesson(j));
    }
   }
  }
```

```
   secretariatProperties.SetLessonsDeclarationsUpdated(true);
  }

  private void CheckDeclaration(LessonsDeclaration declaration) throws
NotValidLessonsDeclarationException
  {
   int declarations = 0;
   for (int i = 0; i < declaration.GetStudiesProgram().Size(); i++)
   {
    for (int j = 0; j < declaration.GetStudiesProgram().GetSixMonth(i).Size(); j++)
    {
     if (declaration.GetStudiesProgram().GetSixMonth(i).GetLesson(j).IsDeclared())
     {
      declarations +=
declaration.GetStudiesProgram().GetSixMonth(i).GetLesson(j).GetHours();
     }
    }
   }
   if (declarations > declaration.GetMaxDeclaredHours())
   {
    throw (new
NotValidLessonsDeclarationException("NotValidLessonsDeclarationException"));
   }
  }

  public void InsertTask(String lesson, Task task)
  {
   Connection MyConn = null;
   try
   {
    String DueDate = new String();
    String DateStarted = new String();
    String DateCompleted = new String();
```

```
    if (task.GetDueDate() != null)
    {
     DueDate = task.GetDueDate().GetSQLDate().toString();
    }
    if (task.GetDateStarted() != null)
    {
     DateStarted = task.GetDateStarted().GetSQLDate().toString();
    }
    if (task.GetDateCompleted() != null)
    {
     DateCompleted = task.GetDateCompleted().GetSQLDate().toString();
    }
    this.LoadPool();
    MyConn = Pool.GetConnection();
    Statement MyStatement = MyConn.createStatement();
    String SQL = "INSERT INTO Tasks (TaskSubject, TaskText, TaskDueDate,
TaskState, TaskPriority, TaskCondision, TaskDateStarted, TaskDateCompleted,
TaskTotalWork, TaskActualWork, TaskIsNew, TaskLessonsID) ";
    SQL = SQL + "VALUES ('" + task.GetSubject().toCharArray() + "','" +
task.GetText().toCharArray() + "','" + DueDate + "'," + task.GetState() + "," +
task.GetPriority() + "," + task.GetCondision() + ",'" + DateStarted + "','" +
DateCompleted + "'," + task.GetTotalWork() + "," + task.GetActualWork() + ", 1," +
listlessons.GetLesson(lesson).GetID() + ")";
    MyStatement.executeUpdate(SQL);
   }
   catch (SQLException sqle)
   {
    System.err.println(sqle.getMessage());
   }
   catch (ClassNotFoundException cnfe)
   {
    System.err.println(cnfe.getMessage());
   }
```

```
  catch (Exception e)
  {
   System.err.println(e.getMessage());
  }
  finally
  {
   try
   {
    if ( MyConn != null )
    {
     Pool.ReleaseConnection(MyConn);
    }
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
  }
  secretariatProperties.SetLessonTasksUpdated(true);
}


public void SetStudiesProgramSixMonth(SixMonth Value)
{
  int lessons;
  Lesson lesson = new Lesson();
  Connection MyConn = null;
  try
  {
   this.LoadPool();
   MyConn = Pool.GetConnection();
   Statement MyStatement = MyConn.createStatement();
```

```
    String SQL = "INSERT INTO SixMonthProgram (ProgramStudiesProgramID,
ProgramSixMonthID) ";
    SQL = SQL + "VALUES (" + Value.GetStudiesProgramID() + "," +
Value.GetID() + ")";
    MyStatement.executeUpdate(SQL);
    lessons = Value.Size();
    for (int i = 0; i < lessons; i++)
    {
     lesson = Value.GetLesson(i);
        this.SetSixMonthLesson(lesson);
    }
   }
   catch (SQLException sqle)
   {
    System.err.println(sqle.getMessage());
   }
   catch (ClassNotFoundException cnfe)
   {
    System.err.println(cnfe.getMessage());
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
   finally
   {
    try
    {
        if ( MyConn != null )
        {
         Pool.ReleaseConnection(MyConn);
        }
    }
```

```
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
  }
}

 public void SetSixMonthLesson(Lesson Value)
 {
  Teacher teacher = new Teacher();
  int SixMonthID = 0;
  Connection MyConn = null;
  try
  {
   this.LoadPool();
   MyConn = Pool.GetConnection();
   Statement MyStatement = MyConn.createStatement();
   String SQL = "Select From SixMonths (SixMonthID) Where Name = '" +
Value.GetSixMonth() + "'";
   ResultSet RS = MyStatement.executeQuery(SQL);
   if (RS.next())
   {
    SixMonthID = RS.getInt("SixMonthID");
   }
   SQL = "INSERT INTO LessonProgram (ProgramSixMonthID,
ProgramLessonID) ";
   SQL = SQL + "VALUES (" + SixMonthID + "," + Value.GetID() + ")";
   MyStatement.executeUpdate(SQL);
   teacher = Value.GetListTeachers().GetTeacher(0);//GetTeacher();
   this.SetLessonTeacher(teacher);
  }
  catch (SQLException sqle)
  {
```

```
    System.err.println(sqle.getMessage());
   }
   catch (ClassNotFoundException cnfe)
   {
    System.err.println(cnfe.getMessage());
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
   finally
   {
    try
    {
        if ( MyConn != null )
        {
         Pool.ReleaseConnection(MyConn);
        }
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
   }
}

public void SetLessonTeacher(Teacher Value)
{
  Connection MyConn = null;
  try
  {
    this.LoadPool();
    MyConn = Pool.GetConnection();
```

```
    Statement MyStatement = MyConn.createStatement();
    String SQL = "INSERT INTO TeacherLessonProgram (ProgramTeacher,
ProgramLesson) ";
    SQL = SQL + "VALUES (" + Value.GetTeacherID() + "," +
Value.GetLessonID() + ")";
    MyStatement.executeUpdate(SQL);
   }
   catch (SQLException sqle)
   {
    System.err.println(sqle.getMessage());
   }
   catch (ClassNotFoundException cnfe)
   {
    System.err.println(cnfe.getMessage());
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
   finally
   {
    try
    {
     if ( MyConn != null )
     {
      Pool.ReleaseConnection(MyConn);
     }
    }
    catch (Exception e)
    {
     System.err.println(e.getMessage());
    }
   }
```

```
 }

 public synchronized void AddLessonStudent(VirtualStudent student, Lesson lesson)
 {
  Connection MyConn = null;
  try
  {
   this.LoadPool();
   MyConn = Pool.GetConnection();
   Statement MyStatement = MyConn.createStatement();
   String SQL = "INSERT INTO StudentLessonProgram (ProgramStudentID,
ProgramLessonID) ";
   SQL = SQL + "VALUES (" + student.GetStudentID() + "," + lesson.GetID() +
")";
   MyStatement.executeUpdate(SQL);
  }
  catch (SQLException sqle)
  {
   System.err.println(sqle.getMessage());
  }
  catch (ClassNotFoundException cnfe)
  {
   System.err.println(cnfe.getMessage());
  }
  catch (Exception e)
  {
   System.err.println(e.getMessage());
  }
  finally
  {
   try
   {
    if ( MyConn != null )
```

```
      {
        Pool.ReleaseConnection(MyConn);
      }
    }
    catch (Exception e)
    {
      System.err.println(e.getMessage());
    }
  }
}

 public synchronized void RemoveLessonStudent(VirtualStudent student, Lesson
lesson)
 {
  Connection MyConn = null;
  try
  {
    this.LoadPool();
    MyConn = Pool.GetConnection();
    Statement MyStatement = MyConn.createStatement();
    String SQL = "DELETE FROM StudentLessonProgram WHERE
ProgramStudentID = " + student.GetStudentID() + " AND ProgramLessonID = " +
lesson.GetID();
    MyStatement.executeUpdate(SQL);
  }
  catch (SQLException sqle)
  {
    System.err.println(sqle.getMessage());
  }
  catch (ClassNotFoundException cnfe)
  {
    System.err.println(cnfe.getMessage());
  }
```

```
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
   finally
   {
    try
    {
     if ( MyConn != null )
     {
      Pool.ReleaseConnection(MyConn);
     }
    }
    catch (Exception e)
    {
     System.err.println(e.getMessage());
    }
   }
  }

  public SixMonth InsertSixmonth(SixMonth Value)
  {
   SixMonth TempSixMonth = new SixMonth();
   long SixMonthID = 0;
   Connection MyConn = null;
   try
   {
    this.LoadPool();
    MyConn = Pool.GetConnection();
    Statement MyStatement = MyConn.createStatement();
    String SQL = "INSERT INTO SixMonths (Name, Maxlessonsnumber)";
    SQL = SQL + "VALUES ('" + Value.GetName() + "' , " +
10/*Value.GetMaxLessons()*/ + ")";
```

```
    MyStatement.executeUpdate(SQL);
    SQL = "SELECT * FROM SixMonths WHERE Name = '" + Value.GetName() +
"'";
    ResultSet RS = MyStatement.executeQuery(SQL);
    SixMonthID = RS.getLong("SixMonthID");
   }
   catch (SQLException sqle)
   {
    System.err.println(sqle.getMessage());
   }
   catch (ClassNotFoundException cnfe)
   {
    System.err.println(cnfe.getMessage());
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
   finally
   {
    try
    {
        if ( MyConn != null )
        {
         Pool.ReleaseConnection(MyConn);
        }
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
   }
   TempSixMonth = Value;
```

```
   TempSixMonth.SetID(SixMonthID);
   return (TempSixMonth);
  }

  public void InsertLesson(Lesson Value)
  {
   Connection MyConn = null;
   try
   {
    this.LoadPool();
    MyConn = Pool.GetConnection();
    Statement MyStatement = MyConn.createStatement();
    String SQL = "INSERT INTO Lessons (LessonName) VALUES ('" +
Value.GetName() + "')";
    MyStatement.executeUpdate(SQL);
    SQL = "SELECT * FROM Section WHERE (((SectionName) = '" +
Value.GetSection() + "'))";
    ResultSet RS = MyStatement.executeQuery(SQL);
    if (RS.next())
    {
        SQL = "UPDATE Lessons SET LessonSection = " +
RS.getLong("SectionID");
        MyStatement.executeUpdate(SQL);
    }
    RS.close();
    this.SetChangeListLessons(true);
   }
   catch (SQLException sqle)
   {
    System.err.println(sqle.getMessage());
   }
   catch (ClassNotFoundException cnfe)
   {
```

```
    System.err.println(cnfe.getMessage());
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
   finally
   {
    try
    {
        if ( MyConn != null )
        {
         Pool.ReleaseConnection(MyConn);
        }
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
   }
}

public String CheckUser(String Value) throws Exception
{
  String Check = new String();
  if ((Value.equals(null)) || (Value.equals("")))
  {
    Check = "Empty";
  }
  else
  {
    Connection MyConn = null;
    try
```

```
    {
        this.LoadPool();
     MyConn = Pool.GetConnection();
     Statement MyStatement = MyConn.createStatement();
        String SQL = "SELECT (Username) FROM Students WHERE Username = '"
+ Value + "'";
        ResultSet RS = MyStatement.executeQuery(SQL);
        if (RS.next())
        {
       Check = RS.getString("Username");
       Check = "Used";
     }
        else
        {
         Check = "NotUsed";
        }
        RS.close();
    }
    catch (SQLException sqle)
    {
        System.err.println(sqle.getMessage());
    }
    catch (ClassNotFoundException cnfe)
    {
     System.err.println(cnfe.getMessage());
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
    finally
    {
        try
```

```
        {
          if (!MyConn.equals(null))
        {
            Pool.ReleaseConnection(MyConn);
        }
         }
         catch (Exception e)
         {
          System.err.println(e.getMessage());
         }
    }
    if (Check.equals("NotUsed"))
    {
        MyConn = null;
        try
        {
         this.LoadPool();
       MyConn = Pool.GetConnection();
       Statement MyStatement = MyConn.createStatement();
       String SQL = "SELECT (Username) FROM Teachers WHERE Username = '"
+ Value + "'";
          ResultSet RS = MyStatement.executeQuery(SQL);
       if (RS.next())
          {
           Check = RS.getString("Username");
        Check = "Used";
       }
          else
          {
           Check = "NotUsed";
          }
       RS.close();
         }
```

```
        catch (SQLException sqle)
        {
          System.err.println(sqle.getMessage());
        }
        catch (ClassNotFoundException cnfe)
        {
          System.err.println(cnfe.getMessage());
        }
        catch (Exception e)
        {
          System.err.println(e.getMessage());
        }
        finally
        {
          try
          {
            if (!MyConn.equals(null))
            {
              Pool.ReleaseConnection(MyConn);
        }
      }
          catch (Exception e)
          {
            System.err.println(e.getMessage());
      }
        }
}
if (Check.equals("NotUsed"))
{
        MyConn = null;
        try
        {
      this.LoadPool();
```

```
        MyConn = Pool.GetConnection();
        Statement MyStatement = MyConn.createStatement();
           String SQL = "SELECT (Username) FROM Administrators WHERE
Username = '" + Value + "'";
           ResultSet RS = MyStatement.executeQuery(SQL);
        if (RS.next())
           {
            Check = RS.getString("Username");
          Check = "Used";
        }
        else
           {
            Check = "NotUsed";
          System.out.print("Error");
        }
        RS.close();
          }
          catch (SQLException sqle)
          {
           System.err.println(sqle.getMessage());
          }
          catch (ClassNotFoundException cnfe)
          {
           System.err.println(cnfe.getMessage());
          }
          catch (Exception e)
          {
           System.err.println(e.getMessage());
          }
          finally
          {
           try
            {
```

```
            if (!MyConn.equals(null))
            {
              Pool.ReleaseConnection(MyConn);
          }
           }
           catch (Exception e)
           {
            System.err.println(e.getMessage());
           }
      }
    }
  }
  return (Check);
 }

 public String IdentifyUser(String Value)
 {
  String Check = new String();
  if ((Value == null) || (Value.equals("")))
  {
   Check = new String("Empty");
  }
  else
  {
   Connection MyConn = null;
   try
   {
    this.LoadPool();
    MyConn = Pool.GetConnection();
    Statement MyStatement = MyConn.createStatement();
       String SQL = "SELECT (Identification) FROM Students WHERE Username
= '" + Value + "'";
       ResultSet RS = MyStatement.executeQuery(SQL);
```

```
 if (RS.next())
    {
     Check = RS.getString("Identification");
 }
    else
    {
     Check = "NotUsed";
 }
    RS.close();
}
catch (SQLException sqle)
{
    System.err.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
{
    System.err.println(cnfe.getMessage());
}
catch (Exception e)
{
    System.err.println(e.getMessage());
}
finally
{
    try
    {
     if ( MyConn != null )
   {
       Pool.ReleaseConnection(MyConn);
     }
    }
    catch (Exception e)
    {
```

```
            System.err.println(e.getMessage());
          }
        }
        if (Check.equals("NotUsed"))
        {
            MyConn = null;
            try
            {
           this.LoadPool();
           MyConn = Pool.GetConnection();
           Statement MyStatement = MyConn.createStatement();
           String SQL = "SELECT (Identification) FROM Teachers WHERE Username =
'" + Value + "'";
              ResultSet RS = MyStatement.executeQuery(SQL);
           if (RS.next())
              {
               Check = RS.getString("Identification");
           }
              else
              {
               Check = "NotUsed";
           }
           RS.close();
            }
            catch (SQLException sqle)
            {
             System.err.println(sqle.getMessage());
            }
            catch (ClassNotFoundException cnfe)
            {
             System.err.println(cnfe.getMessage());
            }
            catch (Exception e)
```

```
        {
          System.err.println(e.getMessage());
        }
      finally
        {
         try
      {
            if (MyConn != null )
            {
              Pool.ReleaseConnection(MyConn);
        }
            }
           catch (Exception e)
            {
             System.err.println(e.getMessage());
      }
        }
    }
    if (Check.equals("NotUsed"))
    {
        MyConn = null;
        try
        {
       this.LoadPool();
       MyConn = Pool.GetConnection();
       Statement MyStatement = MyConn.createStatement();
       String SQL = "SELECT (Identification) FROM Administrators WHERE
Username = '" + Value + "'";
          ResultSet RS = MyStatement.executeQuery(SQL);
       if (RS.next())
       {
        Check = RS.getString("Identification");
       }
```

```
        else
      {
          Check = "NotUsed";
      }
     RS.close();
   }
     catch (SQLException sqle)
     {
      System.err.println(sqle.getMessage());
     }
     catch (ClassNotFoundException cnfe)
     {
      System.err.println(cnfe.getMessage());
     }
     catch (Exception e)
     {
      System.err.println(e.getMessage());
     }
     finally
     {
      try
      {
       if (MyConn != null )
       {
         Pool.ReleaseConnection(MyConn);
     }
    }
      catch (Exception e)
      {
    System.err.println(e.getMessage());
      }
  }
}
```

```
  }
  return (Check);
}

public void LoadListStudents()
{
  VirtualStudent TempStudent;
  Connection MyConn = null;
  try
  {
    this.LoadPool();
    MyConn = this.Pool.GetConnection();
    Statement MyStatement = MyConn.createStatement();
    String SQL = "SELECT * FROM Students;";
    ResultSet RS = MyStatement.executeQuery(SQL);
    liststudents.RemoveAll();
    while (RS.next())
    {
        TempStudent = new VirtualStudent();
        TempStudent.SetStudentID(RS.getLong("StudentID"));
        TempStudent.SetUsername(RS.getString("Username"));
        TempStudent.SetPassword(RS.getString("Password"));
        TempStudent.SetName(RS.getString("Name"));
        TempStudent.SetLastname(RS.getString("Lastname"));
        TempStudent.SetSex(RS.getString("Sex"));
        TempStudent.SetDay(RS.getString("BornDay"));
        TempStudent.SetMonth(RS.getString("BornMonth"));
        TempStudent.SetYear(RS.getString("BornYear"));
        TempStudent.SetAddress(RS.getString("Address"));
        TempStudent.SetCity(RS.getString("City"));
        TempStudent.SetCountry(RS.getString("Country"));
        TempStudent.SetZipcode(RS.getString("Zipcode"));
        TempStudent.SetOccupation(RS.getString("Occupation"));
```

```
        TempStudent.SetPhone(RS.getString("Phone"));
        TempStudent.SetEmail(RS.getString("Email"));
        TempStudent.SetSQuestion(RS.getString("SQuestion"));
        TempStudent.SetSAnswer(RS.getString("SAnswer"));
        this.liststudents.Add(TempStudent);
  }
  RS.close();
}
catch (SQLException sqle)
{
  System.err.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
{
  System.err.println(cnfe.getMessage());
}
catch (Exception e)
{
  System.err.println(e.getMessage());
}
finally
{
  try
  {
       if ( MyConn != null )
       {
         this.Pool.ReleaseConnection(MyConn);
       }
  }
  catch (Exception e)
  {
       System.err.println(e.getMessage());
  }
```

```
  }
}

public void LoadListTeachers()
{
  VirtualTeacher TempTeacher;
  Connection MyConn = null;
  try
  {
    this.LoadPool();
    MyConn = this.Pool.GetConnection();
    Statement MyStatement = MyConn.createStatement();
    String SQL = "SELECT * FROM Teachers";
    ResultSet RS = MyStatement.executeQuery(SQL);
    listteachers.RemoveAll();
    while (RS.next())
    {
        TempTeacher = new VirtualTeacher();
        TempTeacher.SetTeacherID(RS.getLong("TeacherID"));
        TempTeacher.SetUsername(RS.getString("Username"));
        TempTeacher.SetPassword(RS.getString("Password"));
        TempTeacher.SetName(RS.getString("Name"));
        TempTeacher.SetLastname(RS.getString("Lastname"));
        TempTeacher.SetSex(RS.getString("Sex"));
        TempTeacher.SetDay(RS.getString("BornDay"));
        TempTeacher.SetMonth(RS.getString("BornMonth"));
        TempTeacher.SetYear(RS.getString("BornYear"));
        TempTeacher.SetAddress(RS.getString("Address"));
        TempTeacher.SetCity(RS.getString("City"));
        TempTeacher.SetCountry(RS.getString("Country"));
        TempTeacher.SetZipcode(RS.getString("Zipcode"));
        TempTeacher.SetOccupation(RS.getString("Occupation"));
        TempTeacher.SetPhone(RS.getString("Phone"));
```

```
        TempTeacher.SetEmail(RS.getString("Email"));
        TempTeacher.SetSQuestion(RS.getString("SQuestion"));
        TempTeacher.SetSAnswer(RS.getString("SAnswer"));
        this.listteachers.Add(TempTeacher);
  }
  RS.close();
}
catch (SQLException sqle)
{
  System.err.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
{
  System.err.println(cnfe.getMessage());
}
catch (Exception e)
{
  System.err.println(e.getMessage());
}
finally
{
  try
  {
        if ( MyConn != null )
        {
          this.Pool.ReleaseConnection(MyConn);
        }
  }
  catch (Exception e)
  {
        System.err.println(e.getMessage());
  }
}
```

```
}

public void LoadListAdministrators()
{
 VirtualAdministrator TempAdministrator;
 Connection MyConn = null;
 try
 {
  this.LoadPool();
  MyConn  = this.Pool.GetConnection();
  Statement MyStatement = MyConn.createStatement();
  String SQL = "SELECT * FROM Administrators";
  ResultSet RS = MyStatement.executeQuery(SQL);
  listadministrators.RemoveAll();
  while (RS.next())
  {
      TempAdministrator = new VirtualAdministrator();
      TempAdministrator.SetAdministratorID(RS.getLong("AdministratorID"));
      TempAdministrator.SetUsername(RS.getString("Username"));
      TempAdministrator.SetPassword(RS.getString("Password"));
      TempAdministrator.SetName(RS.getString("Name"));
      TempAdministrator.SetLastname(RS.getString("Lastname"));
      TempAdministrator.SetSex(RS.getString("Sex"));
      TempAdministrator.SetDay(RS.getString("BornDay"));
      TempAdministrator.SetMonth(RS.getString("BornMonth"));
      TempAdministrator.SetYear(RS.getString("BornYear"));
      TempAdministrator.SetAddress(RS.getString("Address"));
      TempAdministrator.SetCity(RS.getString("City"));
      TempAdministrator.SetCountry(RS.getString("Country"));
      TempAdministrator.SetZipcode(RS.getString("Zipcode"));
      TempAdministrator.SetOccupation(RS.getString("Occupation"));
      TempAdministrator.SetPhone(RS.getString("Phone"));
      TempAdministrator.SetEmail(RS.getString("Email"));
```

```
            TempAdministrator.SetSQuestion(RS.getString("SQuestion"));
            TempAdministrator.SetSAnswer(RS.getString("SAnswer"));
            this.listadministrators.Add(TempAdministrator);
        }
        RS.close();
    }
    catch (SQLException sqle)
    {
        System.err.println(sqle.getMessage());
    }
    catch (ClassNotFoundException cnfe)
    {
        System.err.println(cnfe.getMessage());
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
    finally
    {
        try
        {
            if ( MyConn != null )
            {
                this.Pool.ReleaseConnection(MyConn);
            }
        }
        catch (Exception e)
        {
            System.err.println(e.getMessage());
        }
    }
}
```

```
 public void LoadListLessons()
 {
  Lesson TempLesson;
  Connection MyConn = null;
  try
  {
   this.LoadPool();
   MyConn = this.Pool.GetConnection();
   Statement MyStatement = MyConn.createStatement();
   String SQL = "SELECT Lessons.LessonID, Lessons.LessonName,
Section.SectionName, Teachers.TeacherID, Teachers.Name, Teachers.Lastname,
Teachers.Email FROM [Section] INNER JOIN ([Lessons] INNER JOIN ([Teachers]
INNER JOIN [TeacherLessonProgram] ON [Teachers].TeacherID =
[TeacherLessonProgram].ProgramTeacher) ON [Lessons].LessonID =
[TeacherLessonProgram].ProgramLesson) ON [Section].SectionID =
[Lessons].LessonSection";
   ResultSet RS = MyStatement.executeQuery(SQL);
   while (RS.next())
   {
    Connection MyConn2 = null;
    Connection MyConn3 = null;
       TempLesson = new Lesson();
    VirtualTeacher TempTeacher = new VirtualTeacher();
       TempLesson.SetID(RS.getLong("LessonID"));
       TempLesson.SetName(RS.getString("LessonName"));
       TempLesson.SetSection(RS.getString("SectionName"));
       TempTeacher.SetTeacherID(RS.getLong("TeacherID"));
       TempTeacher.SetName(RS.getString("Name"));
       TempTeacher.SetLastname(RS.getString("Lastname"));
       TempTeacher.SetEmail(RS.getString("Email"));
       TempLesson.SetTeacher(TempTeacher);
    try
```

```
    {
     MyConn2 = this.Pool.GetConnection();
     Statement MyStatement2 = MyConn2.createStatement();
     String SQLSixMonth = "SELECT SixMonths.Name FROM LessonProgram
INNER JOIN Lessons ON LessonProgram.ProgramLessonID = Lessons.LessonID
INNER JOIN SixMonths ON LessonProgram.ProgramSixMonthID =
SixMonths.SixMonthID ";
     SQLSixMonth = SQLSixMonth + "WHERE (Lessons.LessonID = " +
TempLesson.GetID() + ")";
     ResultSet RSSixMonth = MyStatement2.executeQuery(SQLSixMonth);
     RSSixMonth.next();
     TempLesson.SetSixMonth(RSSixMonth.getString("Name"));
    }
    catch (SQLException sqle)
    {
     System.err.println(sqle.getMessage());
    }
    catch (ClassNotFoundException cnfe)
    {
     System.err.println(cnfe.getMessage());
    }
    catch (Exception e)
    {
     System.err.println(e.getMessage());
    }
    finally
    {
     try
     {
      if ( MyConn2 != null )
      {
       this.Pool.ReleaseConnection(MyConn2);
      }
```

```
        }
        catch (Exception e)
        {
          System.err.println(e.getMessage());
        }
      }
      try
      {
        MyConn3 = this.Pool.GetConnection();
        Statement MyStatement3 = MyConn3.createStatement();
        String SQL3 = "SELECT LinkedLessonsLinkedLesson From LinkedLessons
Where (LinkedLessonsLessonID = " + TempLesson.GetID() + ")";
        ResultSet RS3 = MyStatement3.executeQuery(SQL3);
        while (RS3.next())
        {

TempLesson.AddRequiredLesson(RS3.getString("LinkedLessonsLinkedLesson"));
        }
        RS3.close();
      }
      catch (SQLException sqle)
        {
        System.err.println(sqle.getMessage());
        }
        catch (ClassNotFoundException cnfe)
        {
          System.err.println(cnfe.getMessage());
        }
        catch (Exception e)
        {
          System.err.println(e.getMessage());
      }
        finally
```

```
        {
         try
         {
          if ( MyConn3 != null )
          {
         this.Pool.ReleaseConnection(MyConn3);
        }
         }
         catch (Exception e)
         {
          System.err.println(e.getMessage());
         }
        }
    this.listlessons.Add(TempLesson);
  }
  RS.close();
}
catch (SQLException sqle)
{
  System.err.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
{
  System.err.println(cnfe.getMessage());
}
catch (Exception e)
{
  System.err.println(e.getMessage());
}
finally
{
  try
  {
```

```
      if ( MyConn != null )
      {
          this.Pool.ReleaseConnection(MyConn);
      }
     }
     catch (Exception e)
     {
      System.err.println(e.getMessage());
     }
    }
    for (int i = 0; i < listlessons.Size(); i++)
    {

listlessons.GetLesson(i).SetLessonNotes(lessonNoteManager.LoadLessonNotes(listle
ssons.GetLesson(i).GetName()));
    }
  }

  public void LoadUsersLessons()
  {
    Lesson TempLesson;
    Connection MyConn = null;
    try
    {
     this.LoadPool();
     MyConn = this.Pool.GetConnection();
     Statement MyStatement = MyConn.createStatement();
     String SQL = "SELECT Lessons.LessonID, Lessons.LessonName,
Section.SectionName, Teachers.TeacherID, Teachers.Username, Teachers.Name,
Teachers.Lastname, Teachers.Email FROM [Section] INNER JOIN ([Lessons]
INNER JOIN ([Teachers] INNER JOIN [TeacherLessonProgram] ON
[Teachers].TeacherID = [TeacherLessonProgram].ProgramTeacher) ON
```

```
[Lessons].LessonID = [TeacherLessonProgram].ProgramLesson) ON
[Section].SectionID = [Lessons].LessonSection";
    ResultSet RS = MyStatement.executeQuery(SQL);
    while (RS.next())
    {
     Connection MyConn2 = null;
     Connection MyConn3 = null;
     TempLesson = new Lesson();
     VirtualTeacher TempTeacher = new VirtualTeacher();
     TempLesson.SetID(RS.getLong("LessonID"));
     TempLesson.SetName(RS.getString("LessonName"));
     TempLesson.SetSection(RS.getString("SectionName"));
     TempTeacher.SetTeacherID(RS.getLong("TeacherID"));
     TempTeacher.SetUsername(RS.getString("Username"));
     TempTeacher.SetName(RS.getString("Name"));
     TempTeacher.SetLastname(RS.getString("Lastname"));
     TempTeacher.SetEmail(RS.getString("Email"));
     TempLesson.SetTeacher(TempTeacher);
     try
     {
      MyConn2 = this.Pool.GetConnection();
      Statement MyStatement2 = MyConn2.createStatement();
      String SQLSixMonth = "SELECT SixMonths.Name FROM LessonProgram
INNER JOIN Lessons ON LessonProgram.ProgramLessonID = Lessons.LessonID
INNER JOIN SixMonths ON LessonProgram.ProgramSixMonthID =
SixMonths.SixMonthID ";
      SQLSixMonth = SQLSixMonth + "WHERE (Lessons.LessonID = " +
TempLesson.GetID() + ")";
      ResultSet RSSixMonth = MyStatement2.executeQuery(SQLSixMonth);
      RSSixMonth.next();
      TempLesson.SetSixMonth(RSSixMonth.getString("Name"));
      RSSixMonth.close();
     }
```

```
      catch (SQLException sqle)
      {
       System.err.println(sqle.getMessage());
      }
      catch (ClassNotFoundException cnfe)
      {
       System.err.println(cnfe.getMessage());
      }
      catch (Exception e)
      {
       System.err.println(e.getMessage());
      }
      finally
      {
       try
       {
        if ( MyConn2 != null )
        {
         this.Pool.ReleaseConnection(MyConn2);
        }
       }
       catch (Exception e)
       {
        System.err.println(e.getMessage());
       }
      }
      try
      {
       MyConn3 = this.Pool.GetConnection();
       Statement MyStatement3 = MyConn3.createStatement();
       String SQL3 = "SELECT LinkedLessonsLinkedLesson From LinkedLessons
Where (LinkedLessonsLessonID = " + TempLesson.GetID() + ")";
       ResultSet RS3 = MyStatement3.executeQuery(SQL3);
```

```
  while (RS3.next())
  {

TempLesson.AddRequiredLesson(RS3.getString("LinkedLessonsLinkedLesson"));
  }
  RS3.close();
 }
 catch (SQLException sqle)
 {
  System.err.println(sqle.getMessage());
 }
 catch (ClassNotFoundException cnfe)
 {
  System.err.println(cnfe.getMessage());
 }
 catch (Exception e)
 {
  System.err.println(e.getMessage());
 }
 finally
 {
  try
  {
       if ( MyConn3 != null )
       {
     this.Pool.ReleaseConnection(MyConn3);
       }
  }
  catch (Exception e)
  {
    System.err.println(e.getMessage());
  }
 }
```

```
Connection MyConnStudent = null;
try
{
 MyConnStudent = this.Pool.GetConnection();
 Statement MyStatementStudent = MyConnStudent.createStatement();
 String SQLStudent = "SELECT Students.Username FROM Lessons INNER
JOIN StudentLessonProgram ON Lessons.LessonID =
StudentLessonProgram.ProgramLessonID INNER JOIN Students ON
StudentLessonProgram.ProgramStudentID = Students.StudentID WHERE
Lessons.LessonID = " + TempLesson.GetID();
 ResultSet RSStudent = MyStatementStudent.executeQuery(SQLStudent);
 while (RSStudent.next())
 {

TempLesson.SetStudent(liststudents.GetStudent(RSStudent.getString("Username")));
 }
 RSStudent.close();
}
catch (SQLException sqle)
{
 System.err.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
{
 System.err.println(cnfe.getMessage());
}
catch (Exception e)
{
 System.err.println(e.getMessage());
}
finally
{
 try
```

```
        {
          if ( MyConnStudent != null )
          {
            this.Pool.ReleaseConnection(MyConnStudent);
          }
        }
        catch (Exception e)
        {
            System.err.println(e.getMessage());
        }
      }
      Connection MyConnTask = null;
      try
      {
        MyConnTask = this.Pool.GetConnection();
        Statement MyStatementTask = MyConnTask.createStatement();
        String SQLTask = "SELECT Tasks.TaskID, Tasks.TaskSubject,
Tasks.TaskText, Tasks.TaskDueDate, Tasks.TaskState, Tasks.TaskPriority,
Tasks.TaskCondision, Tasks.TaskDateStarted, Tasks.TaskDateCompleted,
Tasks.TaskTotalWork, Tasks.TaskActualWork, Tasks.TaskIsNew FROM Lessons
INNER JOIN Tasks ON Lessons.LessonID = Tasks.TaskLessonsID WHERE
Lessons.LessonID = " + TempLesson.GetID();
        ResultSet RSTask = MyStatementTask.executeQuery(SQLTask);
        while (RSTask.next())
        {
          Task task = new Task();
          task.SetID(RSTask.getLong("TaskID"));
          task.SetSubject(RSTask.getString("TaskSubject"));
          String Text = RSTask.getString("TaskText");
          if (!(Text.equals("")) && (Text != null))
          {
            task.SetText(Text);
          }
```

```
String DueDate = RSTask.getString("TaskDueDate");
if (!(DueDate.equals("")) && (DueDate != null))
{
 task.SetDueDate(new DateTimer(DueDate));
}
else
{
 task.SetDueDate(null);
}
task.SetState(RSTask.getInt("TaskState"));
task.SetPriority(RSTask.getInt("TaskPriority"));
task.SetCondision(RSTask.getInt("TaskCondision"));
String DateStarted = RSTask.getString("TaskDateStarted");
if (!(DateStarted.equals("")) && (DateStarted != null))
{
 task.SetDateStarted(new DateTimer(DateStarted));
}
else
{
 task.SetDateStarted(null);
}
String DateCompleted = RSTask.getString("TaskDateCompleted");
if (!(DateCompleted.equals("")) && (DateCompleted != null))
{
 task.SetDateCompleted(new DateTimer(DateCompleted));
}
else
{
 task.SetDateCompleted(null);
}
task.SetTotalWork(RSTask.getInt("TaskTotalWork"));
task.SetActualWork(RSTask.getInt("TaskActualWork"));
int state = RSTask.getInt("TaskIsNew");
```

```
    if (state == 1)
    {
      task.SetNew(true);
    }
    else if (state == 0)
    {
      task.SetNew(false);
    }
    TempLesson.AddTask(task);
  }
  RSTask.close();
}
catch (SQLException sqle)
{
  System.err.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
{
  System.err.println(cnfe.getMessage());
}
catch (Exception e)
{
  System.err.println(e.getMessage());
}
finally
{
  try
  {
    if ( MyConnTask != null )
    {
      this.Pool.ReleaseConnection(MyConnTask);
    }
  }
```

```
      catch (Exception e)
      {
           System.err.println(e.getMessage());
      }
    }
    this.lessonTasks.Add(TempLesson);
  }
  RS.close();
}
catch (SQLException sqle)
{
  System.err.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
{
  System.err.println(cnfe.getMessage());
}
catch (Exception e)
{
  System.err.println(e.getMessage());
}
finally
{
  try
  {
    if ( MyConn != null )
    {
         this.Pool.ReleaseConnection(MyConn);
    }
  }
  catch (Exception e)
  {
    System.err.println(e.getMessage());
```

```
    }
   }
   for (int i = 0; i < lessonTasks.Size(); i++)
   {

lessonTasks.GetLesson(i).SetLessonNotes(lessonNoteManager.LoadLessonNotes(less
onTasks.GetLesson(i).GetName()));
   }
  }

  public void LoadDailyProgram()
  {
   DateTimer datetimer = new DateTimer();
   VirtualTeacher TempTeacher;
   Lesson TempLesson;
   Classs TempClass;
   Day day = new Day();
   java.sql.Date ClassDate;
   Connection MyConn = null;
   long classid;
   long classlessonid;
   String sixmonthname;
   String Name;
   String Subject;
   String date;
   String time;
   long teacherid;
   String teachername;
   String teacherlastname;
   String email;
   try
   {
    this.LoadPool();
```

```
MyConn = this.Pool.GetConnection();
Statement MyStatement = MyConn.createStatement();
String SQL = "SELECT Classes.ClassID, Classes.ClassName, Classes.Subject,
Classes.ClassDate, Classes.ClassTime, Classes.ClassLessonID, Teachers.TeacherID,
Teachers.Name, Teachers.Lastname, Teachers.Email FROM Teachers INNER JOIN
(Classes INNER JOIN TeacherClassProgram ON Classes.ClassID =
TeacherClassProgram.ProgramClass) ON Teachers.TeacherID =
TeacherClassProgram.ProgramTeacher";
ResultSet RS = MyStatement.executeQuery(SQL);
while (RS.next())
{
    TempClass = new Classs();
  TempLesson = new Lesson();
    TempTeacher = new VirtualTeacher();
    classid = RS.getLong("ClassID");
    Name = RS.getString("ClassName");
    Subject = RS.getString("Subject");
    date = RS.getString("ClassDate");
    time = RS.getString("ClassTime");
 classlessonid = RS.getLong("ClassLessonID");
    teacherid = RS.getLong("TeacherID");
    teachername = RS.getString("Name");
    teacherlastname = RS.getString("Lastname");
    email = RS.getString("Email");
    ClassDate = java.sql.Date.valueOf(date);
    if (datetimer.GetSQLDate().equals(ClassDate))
    {
      TempClass.SetID(classid);
   TempClass.SetName(Name);
      TempLesson.SetSection(Subject);
      TempClass.SetDate(date);
      TempClass.SetTime(time);
      TempTeacher.SetTeacherID(teacherid);
```

```
            TempTeacher.SetName(teachername);
            TempTeacher.SetLastname(teacherlastname);
            TempTeacher.SetEmail(email);
            TempLesson.SetTeacher(TempTeacher);
        sixmonthname = new String();
        Connection MyConn2 = null;
        try
        {
          MyConn2 = this.Pool.GetConnection();
          Statement MyStatement2 = MyConn2.createStatement();
          String SQLSixMonth = "SELECT SixMonths.Name FROM LessonProgram
INNER JOIN Lessons ON LessonProgram.ProgramLessonID = Lessons.LessonID
INNER JOIN SixMonths ON LessonProgram.ProgramSixMonthID =
SixMonths.SixMonthID ";
          SQLSixMonth = SQLSixMonth + "WHERE (Lessons.LessonID = " +
classlessonid + ")";
          ResultSet RSSixMonth = MyStatement2.executeQuery(SQLSixMonth);
          RSSixMonth.next();
          sixmonthname = RSSixMonth.getString("Name");
        }
        catch (SQLException sqle)
            {
              System.err.println(sqle.getMessage());
        }
            catch (ClassNotFoundException cnfe)
            {
              System.err.println(cnfe.getMessage());
            }
            catch (Exception e)
            {
          System.err.println(e.getMessage());
        }
            finally
```

```
        {
         try
         {
           if ( MyConn2 != null )
        {
                this.Pool.ReleaseConnection(MyConn2);
        }
       }
          catch (Exception e)
          {
           System.err.println(e.getMessage());
       }
        }
     TempLesson.SetSixMonth(sixmonthname);
     TempClass.SetLesson(TempLesson);
        day.SetClass(TempClass);
       }
  }
  day.SetName(datetimer.GetDay());
  day.SetDate(datetimer.GetPrintDate());
  dailyprogram.SetDate(datetimer.GetSQLDate());
  dailyprogram.SetDay(day);
  RS.close();
}
catch (SQLException sqle)
{
  System.err.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
{
  System.err.println(cnfe.getMessage());
}
catch (Exception e)
```

```
  {
   System.err.println(e.getMessage());
  }
  finally
  {
   try
   {
        if ( MyConn != null )
        {
         this.Pool.ReleaseConnection(MyConn);
        }
   }
   catch (Exception e)
   {
        System.err.println(e.getMessage());
   }
  }
}

public DailyProgram LoadDailyProgram(DateTimer Value)
{
  DailyProgram dailyprogram = new DailyProgram();
  VirtualTeacher TempTeacher;
  Lesson TempLesson;
  Classs TempClass;
  Day day = new Day();
  java.sql.Date ClassDate;
  Connection MyConn = null;
  long classid;
  long classlessonid;
  String sixmonthname;
  String Name;
  String Subject;
```

```
String date;
String time;
long teacherid;
String teachername;
String teacherlastname;
String email;
try
{
 this.LoadPool();
 MyConn = this.Pool.GetConnection();
 Statement MyStatement = MyConn.createStatement();
 String SQL = "SELECT Classes.ClassID, Classes.ClassName, Classes.Subject,
Classes.ClassDate, Classes.ClassTime, Classes.ClassLessonID, Teachers.TeacherID,
Teachers.Name, Teachers.Lastname, Teachers.Email FROM Teachers INNER JOIN
(Classes INNER JOIN TeacherClassProgram ON Classes.ClassID =
TeacherClassProgram.ProgramClass) ON Teachers.TeacherID =
TeacherClassProgram.ProgramTeacher";
 ResultSet RS = MyStatement.executeQuery(SQL);
 while (RS.next())
 {
     TempClass = new Classs();
  TempLesson = new Lesson();
     TempTeacher = new VirtualTeacher();
     classid = RS.getLong("ClassID");
     Name = RS.getString("ClassName");
     subject = RS.getString("Subject");
     date = RS.getString("ClassDate");
     time = RS.getString("ClassTime");
   classlessonid = RS.getLong("ClassLessonID");
     teacherid = RS.getLong("TeacherID");
     teachername = RS.getString("Name");
     teacherlastname = RS.getString("Lastname");
     email = RS.getString("Email");
```

```
          ClassDate = java.sql.Date.valueOf(date);
          if (Value.GetSQLDate().equals(ClassDate))
          {
            TempClass.SetID(classid);
            TempClass.SetName(Name);
            TempLesson.SetSection(Subject);
            TempClass.SetDate(date);
            TempClass.SetTime(time);
            TempTeacher.SetTeacherID(teacherid);
            TempTeacher.SetName(teachername);
            TempTeacher.SetLastname(teacherlastname);
            TempTeacher.SetEmail(email);
            TempLesson.SetTeacher(TempTeacher);
        sixmonthname = new String();
        Connection MyConn2 = null;
        try
        {
          MyConn2 = this.Pool.GetConnection();
          Statement MyStatement2 = MyConn2.createStatement();
          String SQLSixMonth = "SELECT SixMonths.Name FROM LessonProgram
INNER JOIN Lessons ON LessonProgram.ProgramLessonID = Lessons.LessonID
INNER JOIN SixMonths ON LessonProgram.ProgramSixMonthID =
SixMonths.SixMonthID ";
          SQLSixMonth = SQLSixMonth + "WHERE (Lessons.LessonID = " +
classlessonid + ")";
          ResultSet RSSixMonth = MyStatement2.executeQuery(SQLSixMonth);
          RSSixMonth.next();
          sixmonthname = RSSixMonth.getString("Name");
        }
        catch (SQLException sqle)
           {
             System.err.println(sqle.getMessage());
           }
```

```
        catch (ClassNotFoundException cnfe)
        {
          System.err.println(cnfe.getMessage());
        }
        catch (Exception e)
        {
       System.err.println(e.getMessage());
     }
        finally
        {
         try
         {
           if ( MyConn2 != null )
           {
             this.Pool.ReleaseConnection(MyConn2);
           }
         }
         catch (Exception e)
         {
          System.err.println(e.getMessage());
       }
        }
     TempLesson.SetSixMonth(sixmonthname);
     TempClass.SetLesson(TempLesson);
        day.SetClass(TempClass);
    }
  }
  day.SetName(Value.GetDay());
  day.SetDate(Value.GetPrintDate());
  dailyprogram.SetDate(Value.GetSQLDate());
  dailyprogram.SetDay(day);
  RS.close();
}
```

```
  catch (SQLException sqle)
  {
   System.err.println(sqle.getMessage());
  }
  catch (ClassNotFoundException cnfe)
  {
   System.err.println(cnfe.getMessage());
  }
  catch (Exception e)
  {
   System.err.println(e.getMessage());
  }
  finally
  {
   try
   {
        if ( MyConn != null )
     {
          this.Pool.ReleaseConnection(MyConn);
        }
   }
   catch (Exception e)
   {
        System.err.println(e.getMessage());
   }
  }
  return (dailyprogram);
 }

 public synchronized DailyProgram LoadTeacherDailyProgram(DateTimer Value,
VirtualTeacher teacher)
 {
  DailyProgram dailyprogram = new DailyProgram();
```

```
    VirtualTeacher TempTeacher;
    Lesson TempLesson;
    Classs TempClass;
    Day day = new Day();
    java.sql.Date ClassDate;
    Connection MyConn = null;
    long classid;
    long classlessonid;
    String sixmonthname;
    String Name;
    String Subject;
    String date;
    String time;
    long teacherid;
    String teachername;
    String teacherlastname;
    String email;
    try
    {
      this.LoadPool();
      MyConn = this.Pool.GetConnection();
      Statement MyStatement = MyConn.createStatement();
      String SQL = "SELECT Classes.ClassID, Classes.ClassName, Classes.Subject,
Classes.ClassDate, Classes.ClassTime, Classes.ClassLessonID, Teachers.TeacherID,
Teachers.Name, Teachers.Lastname, Teachers.Email FROM Teachers INNER JOIN
(Classes INNER JOIN TeacherClassProgram ON Classes.ClassID =
TeacherClassProgram.ProgramClass) ON Teachers.TeacherID =
TeacherClassProgram.ProgramTeacher";
      ResultSet RS = MyStatement.executeQuery(SQL);
      while (RS.next())
      {
          TempClass = new Classs();
        TempLesson = new Lesson();
```

```
            TempTeacher = new VirtualTeacher();
            classid = RS.getLong("ClassID");
            Name = RS.getString("ClassName");
            Subject = RS.getString("Subject");
            date = RS.getString("ClassDate");
            time = RS.getString("ClassTime");
          classlessonid = RS.getLong("ClassLessonID");
            teacherid = RS.getLong("TeacherID");
            teachername = RS.getString("Name");
            teacherlastname = RS.getString("Lastname");
            email = RS.getString("Email");
            ClassDate = java.sql.Date.valueOf(date);
          if ((Value.GetSQLDate().equals(ClassDate)) && teacherid ==
teacher.GetTeacherID())
            {
              TempClass.SetID(classid);
              TempClass.SetName(Name);
              TempLesson.SetSection(Subject);
              TempClass.SetDate(date);
              TempClass.SetTime(time);
              TempTeacher.SetTeacherID(teacherid);
              TempTeacher.SetName(teachername);
              TempTeacher.SetLastname(teacherlastname);
              TempTeacher.SetEmail(email);
              TempLesson.SetTeacher(TempTeacher);
            sixmonthname = new String();
            Connection MyConn2 = null;
            try
            {
              MyConn2 = this.Pool.GetConnection();
              Statement MyStatement2 = MyConn2.createStatement();
              String SQLSixMonth = "SELECT SixMonths.Name FROM LessonProgram
INNER JOIN Lessons ON LessonProgram.ProgramLessonID = Lessons.LessonID
```

```
INNER JOIN SixMonths ON LessonProgram.ProgramSixMonthID =
SixMonths.SixMonthID ";
        SQLSixMonth = SQLSixMonth + "WHERE (Lessons.LessonID = " +
classlessonid + ")";
        ResultSet RSSixMonth = MyStatement2.executeQuery(SQLSixMonth);
        RSSixMonth.next();
        sixmonthname = RSSixMonth.getString("Name");
      }
      catch (SQLException sqle)
        {
         System.err.println(sqle.getMessage());
        }
        catch (ClassNotFoundException cnfe)
        {
         System.err.println(cnfe.getMessage());
        }
        catch (Exception e)
        {
      System.err.println(e.getMessage());
      }
        finally
        {
         try
         {
           if ( MyConn2 != null )
        {
         this.Pool.ReleaseConnection(MyConn2);
        }
      }
         catch (Exception e)
         {
           System.err.println(e.getMessage());
      }
```

```
            }
        TempLesson.SetSixMonth(sixmonthname);
        TempClass.SetLesson(TempLesson);
            day.SetClass(TempClass);
      }
    }
    day.SetName(Value.GetDay());
    day.SetDate(Value.GetPrintDate());
    dailyprogram.SetDate(Value.GetSQLDate());
    dailyprogram.SetDay(day);
    RS.close();
  }
  catch (SQLException sqle)
  {
    System.err.println(sqle.getMessage());
  }
  catch (ClassNotFoundException cnfe)
  {
    System.err.println(cnfe.getMessage());
  }
  catch (Exception e)
  {
    System.err.println(e.getMessage());
  }
  finally
  {
    try
    {
        if ( MyConn != null )
        {
          this.Pool.ReleaseConnection(MyConn);
        }
    }
```

```
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
  }
  return (dailyprogram);
 }

 public synchronized WeeklyProgram LoadTeacherWeeklyProgram(DateTimer
Value, VirtualTeacher teacher)
 {
  WeeklyProgram weeklyprogram = new WeeklyProgram();
  DateTimer Weekdatetimer;
  DailyProgram TempDailyProgram;
  java.sql.Timestamp date;
  int Year = Value.GetIntYear();
  int Month = Value.GetIntMonth();
  int Dayofweek = Value.GetIntDayOfWeek();
  int Dayofmonth = Value.GetIntDayOfMonth();
  int FirstDay = (Dayofmonth - Dayofweek);
  date = java.sql.Timestamp.valueOf(Year + "-" + Month + "-" + FirstDay + "
00:00:00.000000001");
  Weekdatetimer = new DateTimer(date);
  for (int i = 0; i < 7; i++)
  {
   Weekdatetimer.SetNextDay();
   TempDailyProgram = new DailyProgram();
   TempDailyProgram = this.LoadTeacherDailyProgram(Weekdatetimer, teacher);
   TempDailyProgram.LoadDailyProgram();
   weeklyprogram.SetDailyProgram(TempDailyProgram);
  }
  weeklyprogram.SetUser(teacher.GetUsername());
  weeklyprogram.SetDate(Value.GetSQLDate());
```

```
    weeklyprogram.SetToday(Dayofweek - 1);
    return (weeklyprogram);
  }

  public synchronized DailyProgram LoadStudentDailyProgram(DateTimer Value,
VirtualStudent student)
  {
    DailyProgram dailyprogram = new DailyProgram();
    VirtualTeacher TempTeacher;
    Lesson TempLesson;
    Classs TempClass;
    Day day = new Day();
    java.sql.Date ClassDate;
    Connection MyConn = null;
    long classid;
    long classlessonid;
    String sixmonthname;
    String Name;
    String Subject;
    String date;
    String time;
    long teacherid;
    String teachername;
    String teacherlastname;
    String email;
    try
    {
      this.LoadPool();
      MyConn = this.Pool.GetConnection();
      Statement MyStatement = MyConn.createStatement();
      String SQL = "SELECT Classes.ClassID, Classes.ClassName, Classes.Subject,
Classes.ClassDate, Classes.ClassTime, Classes.ClassLessonID, Teachers.TeacherID,
Teachers.Name, Teachers.Lastname, Teachers.Email FROM Teachers INNER JOIN
```

```
(Classes INNER JOIN TeacherClassProgram ON Classes.ClassID =
TeacherClassProgram.ProgramClass) ON Teachers.TeacherID =
TeacherClassProgram.ProgramTeacher";
    ResultSet RS = MyStatement.executeQuery(SQL);
    while (RS.next())
    {
     TempClass = new Classs();
     TempLesson = new Lesson();
        TempTeacher = new VirtualTeacher();
        classid = RS.getLong("ClassID");
        Name = RS.getString("ClassName");
        Subject = RS.getString("Subject");
        date = RS.getString("ClassDate");
        time = RS.getString("ClassTime");
     classlessonid = RS.getLong("ClassLessonID");
        teacherid = RS.getLong("TeacherID");
        teachername = RS.getString("Name");
        teacherlastname = RS.getString("Lastname");
        email = RS.getString("Email");
        ClassDate = java.sql.Date.valueOf(date);
     if (Value.GetSQLDate().equals(ClassDate))
     {
      TempClass.SetID(classid);
      TempClass.SetName(Name);
      TempLesson.SetID(classlessonid);
      TempLesson.SetSection(Subject);
         TempClass.SetDate(date);
         TempClass.SetTime(time);
      TempTeacher.SetTeacherID(teacherid);
         TempTeacher.SetName(teachername);
         TempTeacher.SetLastname(teacherlastname);
         TempTeacher.SetEmail(email);
         TempLesson.SetTeacher(TempTeacher);
```

```
sixmonthname = new String();
Connection MyConn3 = null;
try
{
 MyConn3 = this.Pool.GetConnection();
 Statement MyStatement3 = MyConn3.createStatement();
 String findSQL = "SELECT Students.Username, Lessons.LessonID FROM
Lessons INNER JOIN StudentLessonProgram ON dbo.Lessons.LessonID =
dbo.StudentLessonProgram.ProgramLessonID INNER JOIN Students ON
StudentLessonProgram.ProgramStudentID = Students.StudentID WHERE
(Students.Username = '" + student.GetUsername() + "') AND (Lessons.LessonID = "
+ TempLesson.GetID() + ")";
 ResultSet result = MyStatement3.executeQuery(findSQL);
 if (result.next())
 {
  String tmpName = result.getString("Username");
  long tmpID = result.getLong("LessonID");
  Connection MyConn2 = null;
  try
  {
   MyConn2 = this.Pool.GetConnection();
   Statement MyStatement2 = MyConn2.createStatement();
   String SQLSixMonth = "SELECT SixMonths.Name FROM LessonProgram
INNER JOIN Lessons ON LessonProgram.ProgramLessonID = Lessons.LessonID
INNER JOIN SixMonths ON LessonProgram.ProgramSixMonthID =
SixMonths.SixMonthID ";
   SQLSixMonth = SQLSixMonth + "WHERE (Lessons.LessonID = " +
classlessonid + ")";
   ResultSet RSSixMonth = MyStatement2.executeQuery(SQLSixMonth);
   RSSixMonth.next();
   sixmonthname = RSSixMonth.getString("Name");
   TempLesson.SetSixMonth(sixmonthname);
   TempClass.SetLesson(TempLesson);
```

```
            day.SetClass(TempClass);
        }
        catch (SQLException sqle)
           {
            System.err.println(sqle.getMessage());
           }
           catch (ClassNotFoundException cnfe)
           {
            System.err.println(cnfe.getMessage());
           }
           catch (Exception e)
           {
         System.err.println(e.getMessage());
        }
           finally
           {
            try
            {
              if ( MyConn2 != null )
           {
                    this.Pool.ReleaseConnection(MyConn2);
           }
         }
           catch (Exception e)
           {
            System.err.println(e.getMessage());
         }
        }
      }
    }
    catch (SQLException sqle)
       {
        System.err.println(sqle.getMessage());
```

```
      }
         catch (ClassNotFoundException cnfe)
         {
          System.err.println(cnfe.getMessage());
      }
         catch (Exception e)
         {
        System.err.println(e.getMessage());
      }
         finally
         {
          try
          {
            if ( MyConn3 != null )
         {
                this.Pool.ReleaseConnection(MyConn3);
         }
        }
         catch (Exception e)
          {
           System.err.println(e.getMessage());
       }
      }
    }
  }
  day.SetName(Value.GetDay());
  day.SetDate(Value.GetPrintDate());
  dailyprogram.SetDate(Value.GetSQLDate());
  dailyprogram.SetDay(day);
  RS.close();
}
catch (SQLException sqle)
{
```

```
    System.err.println(sqle.getMessage());
   }
   catch (ClassNotFoundException cnfe)
   {
    System.err.println(cnfe.getMessage());
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
   finally
   {
    try
    {
        if ( MyConn != null )
        {
         this.Pool.ReleaseConnection(MyConn);
        }
    }
    catch (Exception e)
    {
     System.err.println(e.getMessage());
    }
   }
   return (dailyprogram);
  }

  public synchronized WeeklyProgram LoadStudentWeeklyProgram(DateTimer
Value, VirtualStudent student)
  {
   WeeklyProgram weeklyprogram = new WeeklyProgram();
   DateTimer Weekdatetimer;
   DailyProgram TempDailyProgram;
```

```
   java.sql.Timestamp date;
   int Year = Value.GetIntYear();
   int Month = Value.GetIntMonth();
   int Dayofweek = Value.GetIntDayOfWeek();
   int Dayofmonth = Value.GetIntDayOfMonth();
   int FirstDay = (Dayofmonth - Dayofweek);
   date = java.sql.Timestamp.valueOf(Year + "-" + Month + "-" + FirstDay + "
00:00:00.000000001");
   Weekdatetimer = new DateTimer(date);
   for (int i = 0; i < 7; i++)
   {
    Weekdatetimer.SetNextDay();
    TempDailyProgram = new DailyProgram();
    TempDailyProgram = this.LoadStudentDailyProgram(Weekdatetimer, student);
    TempDailyProgram.LoadDailyProgram();
    weeklyprogram.SetDailyProgram(TempDailyProgram);
   }
   weeklyprogram.SetUser(student.GetUsername());
   weeklyprogram.SetDate(Value.GetSQLDate());
   weeklyprogram.SetToday(Dayofweek - 1);
   return (weeklyprogram);
  }

  public synchronized void LoadTeachersPrograms()
  {
   synchronized (teachersPrograms)
   {
    teachersPrograms = new ListWeeklyPrograms();
    for (int i = 0; i < listteachers.Size(); i++)
    {
     teachersPrograms.Add(LoadTeacherWeeklyProgram(new DateTimer(),
listteachers.GetTeacher(i)));
    }
```

```
    }
  }

  public synchronized void LoadStudentsPrograms()
  {
    synchronized (studentsPrograms)
    {
      studentsPrograms = new ListWeeklyPrograms();
      for (int i = 0; i < liststudents.Size(); i++)
      {
        studentsPrograms.Add(LoadStudentWeeklyProgram(new DateTimer(),
liststudents.GetStudent(i)));
      }
    }
  }

  public void LoadWeeklyProgram()
  {
    DateTimer datetimer = new DateTimer();
    DateTimer Weekdatetimer;
    DailyProgram TempDailyProgram;
    java.sql.Timestamp date;
    int Year = datetimer.GetIntYear();
    int Month = datetimer.GetIntMonth();
    int Dayofweek = datetimer.GetIntDayOfWeek();
    int Dayofmonth = datetimer.GetIntDayOfMonth();
    int FirstDay = (Dayofmonth - Dayofweek);
    date = java.sql.Timestamp.valueOf(Year + "-" + Month + "-" + FirstDay + "
00:00:00.000000001");
    Weekdatetimer = new DateTimer(date);
    for (int i = 0; i < 7; i++)
    {
      Weekdatetimer.SetNextDay();
```

```
      TempDailyProgram = new DailyProgram();
      TempDailyProgram = this.LoadDailyProgram(Weekdatetimer);
      TempDailyProgram.LoadDailyProgram();
      weeklyprogram.SetDailyProgram(TempDailyProgram);
    }
    weeklyprogram.SetDate(datetimer.GetSQLDate());
    weeklyprogram.SetToday(Dayofweek - 1);
  }

  public WeeklyProgram LoadWeeklyProgram(DateTimer Value)
  {
    WeeklyProgram weeklyprogram = new WeeklyProgram();
    DateTimer Weekdatetimer;
    DailyProgram TempDailyProgram;
    java.sql.Timestamp date;
    int Year = Value.GetIntYear();
    int Month = Value.GetIntMonth();
    int Dayofweek = Value.GetIntDayOfWeek();
    int Dayofmonth = Value.GetIntDayOfMonth();
    int FirstDay = (Dayofmonth - Dayofweek);
    date = java.sql.Timestamp.valueOf(Year + "-" + Month + "-" + FirstDay + "
00:00:00.000000001");
    Weekdatetimer = new DateTimer(date);
    for (int i = 0; i < 7; i++)
    {
      Weekdatetimer.SetNextDay();
      TempDailyProgram = new DailyProgram();
      TempDailyProgram = this.LoadDailyProgram(Weekdatetimer);
      TempDailyProgram.LoadDailyProgram();
      weeklyprogram.SetDailyProgram(TempDailyProgram);
    }
    weeklyprogram.SetDate(Value.GetSQLDate());
    weeklyprogram.SetToday(Dayofweek - 1);
```

```
    return (weeklyprogram);
  }
}
```

Β. Αντικείμενο University

```
package learningclass.*;

import java.sql.*;
import java.net.InetAddress;
import java.util.Vector;
import learningclass.tools.database.ConnectionPool;
import learningclass.users.*;
import learningclass.documents.*;
import learningclass.lessons.*;

public class University
{
  private ListStudents connectedStudents = new ListStudents();
  private ListTeachers connectedTeachers = new ListTeachers();
  private ListAdministrators connectedAdministrators = new ListAdministrators();
  private Principal principal = new Principal();
  public static Secretariat secretariat = new Secretariat();
  public static StudiesProgram studiesprogram = new StudiesProgram();

  public University ()
  {
    try
    {
      studiesprogram = this.LoadStudiesProgram(1);
    }
    catch (Exception e)
    {
```

```
    System.out.println(e.getMessage());
  }
}

public Secretariat GetSecretariat()
{
 return (secretariat);
}

public synchronized int AddConnectedStudent(VirtualStudent Value)
{
 connectedStudents.Add(Value);
 return (connectedStudents.IndexOf(Value));
}

public synchronized int AddConnectedTeacher(VirtualTeacher Value)
{
 connectedTeachers.Add(Value);
 return (connectedTeachers.IndexOf(Value));
}

public synchronized void AddConnectedPrincipal(Principal Value)
{
 principal = Value;
}

public synchronized int AddConnectedAdministrator(VirtualAdministrator Value)
{
 connectedAdministrators.Add(Value);
 return (connectedAdministrators.IndexOf(Value));
}

public synchronized void RemoveConnectedStudent(int Value)
```

```
{
  connectedStudents.Update(null, Value);
}

public synchronized void RemoveConnectedTeacher(int Value)
{
  connectedTeachers.Update(null, Value);
}

public synchronized void RemoveConnectedPrincipal()
{
  principal = null;
}

public synchronized void RemoveConnectedAdministrator(int Value)
{
  connectedAdministrators.Update(null, Value);
}

public int GetConnectedStudentsSize()
{
  return (connectedStudents.SizeInUse());
}

public int GetConnectedTeachersSize()
{
  return (connectedTeachers.SizeInUse());
}

public int GetConnectedAdministratorsSize()
{
  return (connectedAdministrators.SizeInUse());
}
```

```
public Principal GetConnectedPrincipal()
{
 return (principal);
}

public VirtualStudent GetConnectedStudent(int Value)
{
 return (connectedStudents.GetStudent(Value));
}

public VirtualTeacher GetConnectedTeacher(int Value)
{
 return (connectedTeachers.GetTeacher(Value));
}

public VirtualAdministrator GetConnectedAdministrator(int Value)
{
 return (connectedAdministrators.GetAdministrator(Value));
}
public ListStudents GetConnectedStudents()
{
 ListStudents tempList = new ListStudents();
 for (int i = 0; i < connectedStudents.Size(); i++)
 {
  if (connectedStudents.GetStudent(i) != null)
  {
   tempList.Add(connectedStudents.GetStudent(i));
  }
 }
 return (tempList);
}
```

```
public ListTeachers GetConnectedTeachers()
{
 ListTeachers tempList = new ListTeachers();
 for (int i = 0; i < connectedTeachers.Size(); i++)
 {
  if (connectedTeachers.GetTeacher(i) != null)
  {
   tempList.Add(connectedTeachers.GetTeacher(i));
  }
 }
 return (tempList);
}

public ListAdministrators GetConnectedAdministrators()
{
 ListAdministrators tempList = new ListAdministrators();
 for (int i = 0; i < connectedAdministrators.Size(); i++)
 {
  if (connectedAdministrators.GetAdministrator(i) != null)
  {
   tempList.Add(connectedAdministrators.GetAdministrator(i));
  }
 }
 return (tempList);
}

public StudiesProgram GetStudiesProgram()
{
 return (studiesprogram);
}

public boolean IsUserConnected(User user)
{
```

```
  if ((connectedAdministrators.Contains(user)) ||
      (connectedStudents.Contains(user)) ||
      (connectedTeachers.Contains(user)) ||
      (user.equals((User)principal)))
  {
    return (true);
  }
  else
  {
    return (false);
  }
}

public boolean IsUserConnected(String user)
{
  if ((connectedAdministrators.Contains(user)) ||
      (connectedStudents.Contains(user)) ||
      (connectedTeachers.Contains(user)) ||
      (user.equals((User)principal)))
  {
    return (true);
  }
  else
  {
    return (false);
  }
}

public synchronized int Login(User user)
{
  int Status = -1;
  String Check = new String("NotUsed");
  String CheckUser = new String();
```

```
String CheckIdentify = new String();
try
{
 CheckUser = secretariat.CheckUser(user.GetUsername());
}
catch (Exception e)
{
 System.err.println(e.getMessage());
}
CheckIdentify = secretariat.IdentifyUser(user.GetUsername());
if ((CheckUser.equals("Used")) && (CheckIdentify.equals("Student")))
{
 Connection MyConn = null;
 try
 {
  this.secretariat.LoadPool();
     MyConn = this.secretariat.GetConnection();
     Statement MyStatement = MyConn.createStatement();
     String SQL = "SELECT * FROM Students WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "' AND Logedin
= 0 ";
     ResultSet RS = MyStatement.executeQuery(SQL);
     if (RS.next())
     {
    VirtualStudent TempStudent = new VirtualStudent();
      TempStudent.SetStudentID(RS.getLong("StudentID"));
    TempStudent.SetIP(user.GetIP());
      TempStudent.SetUsername(RS.getString("Username"));
    TempStudent.SetPassword(RS.getString("Password"));
      TempStudent.SetName(RS.getString("Name"));
      TempStudent.SetLastname(RS.getString("Lastname"));
      TempStudent.SetSex(RS.getString("Sex"));
      TempStudent.SetDay(RS.getString("BornDay"));
```

```
        TempStudent.SetMonth(RS.getString("BornMonth"));
          TempStudent.SetYear(RS.getString("BornYear"));
          TempStudent.SetAddress(RS.getString("Address"));
          TempStudent.SetCity(RS.getString("City"));
          TempStudent.SetCountry(RS.getString("Country"));
          TempStudent.SetZipcode(RS.getString("Zipcode"));
          TempStudent.SetOccupation(RS.getString("Occupation"));
          TempStudent.SetPhone(RS.getString("Phone"));
          TempStudent.SetEmail(RS.getString("Email"));
          TempStudent.SetSQuestion(RS.getString("SQuestion"));
          TempStudent.SetSAnswer(RS.getString("SAnswer"));
          Status = this.AddConnectedStudent(TempStudent);
        SQL = "Update Students SET Logedin = 1 WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "'";
          MyStatement.executeUpdate(SQL);
          Check = "Login";
        }
        else
        {
         Check = "WrongPassword";
        }
    }
    catch (SQLException sqle)
    {
     System.err.println(sqle.getMessage());
    }
    catch (ClassNotFoundException cnfe)
    {
     System.err.println(cnfe.getMessage());
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
```

```
    }
    finally
    {
     try
     {
         if ( MyConn != null )
       {
           this.secretariat.ReleaseConnection(MyConn);
         }
        }
        catch (Exception e)
        {
         System.err.println(e.getMessage());
        }
    }
   }
   if ((CheckUser.equals("Used")) && (CheckIdentify.equals("Teacher")))
   {
    Connection MyConn = null;
    try
    {
        this.secretariat.LoadPool();
     MyConn = this.secretariat.GetConnection();
     Statement MyStatement = MyConn.createStatement();
        String SQL = "SELECT * FROM Teachers WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "' AND Logedin
= 0 ";
        ResultSet RS = MyStatement.executeQuery(SQL);
     if (RS.next())
        {
         VirtualTeacher TempTeacher = new VirtualTeacher();
       TempTeacher.SetTeacherID(RS.getLong("TeacherID"));
       TempTeacher.SetIP(user.GetIP());
```

```
            TempTeacher.SetUsername(RS.getString("Username"));
            TempTeacher.SetPassword(RS.getString("Password"));
            TempTeacher.SetName(RS.getString("Name"));
            TempTeacher.SetLastname(RS.getString("Lastname"));
            TempTeacher.SetSex(RS.getString("Sex"));
            TempTeacher.SetDay(RS.getString("BornDay"));
            TempTeacher.SetMonth(RS.getString("BornMonth"));
            TempTeacher.SetYear(RS.getString("BornYear"));
            TempTeacher.SetAddress(RS.getString("Address"));
            TempTeacher.SetCity(RS.getString("City"));
            TempTeacher.SetCountry(RS.getString("Country"));
            TempTeacher.SetZipcode(RS.getString("Zipcode"));
            TempTeacher.SetOccupation(RS.getString("Occupation"));
            TempTeacher.SetPhone(RS.getString("Phone"));
            TempTeacher.SetEmail(RS.getString("Email"));
            TempTeacher.SetSQuestion(RS.getString("SQuestion"));
            TempTeacher.SetSAnswer(RS.getString("SAnswer"));
            Status = this.AddConnectedTeacher(TempTeacher);
          SQL = "Update Teachers SET Logedin = 1 WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "'";
            MyStatement.executeUpdate(SQL);
            Check = "Login";
           }
           else
           {
            Check = "WrongPassword";
     }
    }
    catch (SQLException sqle)
    {
        System.err.println(sqle.getMessage());
    }
    catch (ClassNotFoundException cnfe)
```

```
    {
        System.err.println(cnfe.getMessage());
    }
    catch (Exception e)
    {
     System.err.println(e.getMessage());
    }
    finally
    {
        try
        {
         if ( MyConn != null )
         {
           this.secretariat.ReleaseConnection(MyConn);
         }
        }
        catch (Exception e)
        {
         System.err.println(e.getMessage());
     }
    }
   }
   if ((CheckUser.equals("Used")) && (CheckIdentify.equals("Principal")))
   {
    Connection MyConn = null;
    try
    {
     this.secretariat.LoadPool();
        MyConn = this.secretariat.GetConnection();
     Statement MyStatement = MyConn.createStatement();
        String SQL = "SELECT * FROM Teachers WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "' AND Logedin
= 0 ";
```

```
          ResultSet RS = MyStatement.executeQuery(SQL);
          if (RS.next())
          {
        Principal TempPrincipal = new Principal();
            TempPrincipal.SetTeacherID(RS.getLong("TeacherID"));
        TempPrincipal.SetIP(user.GetIP());
            TempPrincipal.SetUsername(RS.getString("Username"));
            TempPrincipal.SetPassword(RS.getString("Password"));
            TempPrincipal.SetName(RS.getString("Name"));
            TempPrincipal.SetLastname(RS.getString("Lastname"));
            TempPrincipal.SetSex(RS.getString("Sex"));
            TempPrincipal.SetDay(RS.getString("BornDay"));
            TempPrincipal.SetMonth(RS.getString("BornMonth"));
            TempPrincipal.SetYear(RS.getString("BornYear"));
            TempPrincipal.SetAddress(RS.getString("Address"));
            TempPrincipal.SetCity(RS.getString("City"));
            TempPrincipal.SetCountry(RS.getString("Country"));
            TempPrincipal.SetZipcode(RS.getString("Zipcode"));
            TempPrincipal.SetOccupation(RS.getString("Occupation"));
            TempPrincipal.SetPhone(RS.getString("Phone"));
            TempPrincipal.SetEmail(RS.getString("Email"));
            TempPrincipal.SetSQuestion(RS.getString("SQuestion"));
            TempPrincipal.SetSAnswer(RS.getString("SAnswer"));
            this.AddConnectedPrincipal(TempPrincipal);
        SQL = "Update Teachers SET Logedin = 1 WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "'";
            MyStatement.executeUpdate(SQL);
            Check = "Login";
          }
          else
          {
            Check = "WrongPassword";
       }
```

```
    }
    catch (SQLException sqle)
    {
        System.err.println(sqle.getMessage());
    }
    catch (ClassNotFoundException cnfe)
    {
        System.err.println(cnfe.getMessage());
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
    finally
    {
        try
        {
          if ( MyConn != null )
       {
            this.secretariat.ReleaseConnection(MyConn);
       }
     }
        catch (Exception e)
        {
          System.err.println(e.getMessage());
        }
    }
}
if ((CheckUser.equals("Used")) && (CheckIdentify.equals("Administrator")))
{
  Connection MyConn = null;
  try
  {
```

```
            this.secretariat.LoadPool();
        MyConn = this.secretariat.GetConnection();
            Statement MyStatement = MyConn.createStatement();
            String SQL = "SELECT * FROM Administrators WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "' AND Logedin
= 0 ";
        ResultSet RS = MyStatement.executeQuery(SQL);
        if (RS.next())
            {
          VirtualAdministrator TempAdministrator = new VirtualAdministrator();
            TempAdministrator.SetAdministratorID(RS.getLong("AdministratorID"));
          TempAdministrator.SetIP(user.GetIP());
            TempAdministrator.SetUsername(RS.getString("Username"));
            TempAdministrator.SetPassword(RS.getString("Password"));
            TempAdministrator.SetName(RS.getString("Name"));
            TempAdministrator.SetLastname(RS.getString("Lastname"));
            TempAdministrator.SetSex(RS.getString("Sex"));
            TempAdministrator.SetDay(RS.getString("BornDay"));
            TempAdministrator.SetMonth(RS.getString("BornMonth"));
            TempAdministrator.SetYear(RS.getString("BornYear"));
            TempAdministrator.SetAddress(RS.getString("Address"));
            TempAdministrator.SetCity(RS.getString("City"));
            TempAdministrator.SetCountry(RS.getString("Country"));
            TempAdministrator.SetZipcode(RS.getString("Zipcode"));
            TempAdministrator.SetOccupation(RS.getString("Occupation"));
            TempAdministrator.SetPhone(RS.getString("Phone"));
            TempAdministrator.SetEmail(RS.getString("Email"));
            TempAdministrator.SetSQuestion(RS.getString("SQuestion"));
            TempAdministrator.SetSAnswer(RS.getString("SAnswer"));
            Status = this.AddConnectedAdministrator(TempAdministrator);
          SQL = "Update Administrators SET Logedin = 1 WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "'";
            MyStatement.executeUpdate(SQL);
```

```
        Check = "Login";
      }
      else
      {
        Check = "WrongPassword";
      }
}
catch (SQLException sqle)
{
  System.err.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
{
      System.err.println(cnfe.getMessage());
}
catch (Exception e)
{
      System.err.println(e.getMessage());
}
finally
{
      try
   {
        if ( MyConn != null )
        {
          this.secretariat.ReleaseConnection(MyConn);
        }
   }
      catch (Exception e)
      {
        System.err.println(e.getMessage());
      }
}
```

```
   }
   return (Status);
  }

  public synchronized int LoginStudent(User user)
  {
   int Status = -1;
   String Check = new String("NotUsed");
   String CheckUser = new String();
   String CheckIdentify = new String();
   try
   {
    CheckUser = secretariat.CheckUser(user.GetUsername());
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
   CheckIdentify = secretariat.IdentifyUser(user.GetUsername());
   if ((CheckUser.equals("Used")) && (CheckIdentify.equals("Student")))
   {
    Connection MyConn = null;
    try
    {
     this.secretariat.LoadPool();
        MyConn = this.secretariat.GetConnection();
        Statement MyStatement = MyConn.createStatement();
        String SQL = "SELECT * FROM Students WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "' AND Logedin
= 0 ";
        ResultSet RS = MyStatement.executeQuery(SQL);
        if (RS.next())
        {
```

```
        VirtualStudent TempStudent = new VirtualStudent();
           TempStudent.SetStudentID(RS.getLong("StudentID"));
        TempStudent.SetIP(user.GetIP());
           TempStudent.SetUsername(RS.getString("Username"));
        TempStudent.SetPassword(RS.getString("Password"));
           TempStudent.SetName(RS.getString("Name"));
           TempStudent.SetLastname(RS.getString("Lastname"));
           TempStudent.SetSex(RS.getString("Sex"));
           TempStudent.SetDay(RS.getString("BornDay"));
        TempStudent.SetMonth(RS.getString("BornMonth"));
           TempStudent.SetYear(RS.getString("BornYear"));
           TempStudent.SetAddress(RS.getString("Address"));
           TempStudent.SetCity(RS.getString("City"));
           TempStudent.SetCountry(RS.getString("Country"));
           TempStudent.SetZipcode(RS.getString("Zipcode"));
           TempStudent.SetOccupation(RS.getString("Occupation"));
           TempStudent.SetPhone(RS.getString("Phone"));
           TempStudent.SetEmail(RS.getString("Email"));
           TempStudent.SetSQuestion(RS.getString("SQuestion"));
           TempStudent.SetSAnswer(RS.getString("SAnswer"));
           Status = this.AddConnectedStudent(TempStudent);
        SQL = "Update Students SET Logedin = 1 WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "'";
           MyStatement.executeUpdate(SQL);
           Check = "Login";
          }
          else
          {
           Check = "WrongPassword";
          }
    }
    catch (SQLException sqle)
    {
```

```
      System.err.println(sqle.getMessage());
    }
    catch (ClassNotFoundException cnfe)
    {
      System.err.println(cnfe.getMessage());
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
    finally
    {
     try
     {
         if ( MyConn != null )
       {
           this.secretariat.ReleaseConnection(MyConn);
         }
        }
        catch (Exception e)
        {
         System.err.println(e.getMessage());
        }
    }
  }
  return (Status);
}

public synchronized int LoginTeacher(User user)
{
  int Status = -1;
  String Check = new String("NotUsed");
  String CheckUser = new String();
```

```
String CheckIdentify = new String();
try
{
 CheckUser = secretariat.CheckUser(user.GetUsername());
}
catch (Exception e)
{
 System.err.println(e.getMessage());
}
CheckIdentify = secretariat.IdentifyUser(user.GetUsername());
if ((CheckUser.equals("Used")) && (CheckIdentify.equals("Teacher")))
{
 Connection MyConn = null;
 try
 {
     this.secretariat.LoadPool();
  MyConn = this.secretariat.GetConnection();
  Statement MyStatement = MyConn.createStatement();
     String SQL = "SELECT * FROM Teachers WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "' AND Logedin
= 0 ";
     ResultSet RS = MyStatement.executeQuery(SQL);
  if (RS.next())
     {
      VirtualTeacher TempTeacher = new VirtualTeacher();
    TempTeacher.SetTeacherID(RS.getLong("TeacherID"));
    TempTeacher.SetIP(user.GetIP());
       TempTeacher.SetUsername(RS.getString("Username"));
       TempTeacher.SetPassword(RS.getString("Password"));
       TempTeacher.SetName(RS.getString("Name"));
       TempTeacher.SetLastname(RS.getString("Lastname"));
       TempTeacher.SetSex(RS.getString("Sex"));
       TempTeacher.SetDay(RS.getString("BornDay"));
```

```
            TempTeacher.SetMonth(RS.getString("BornMonth"));
            TempTeacher.SetYear(RS.getString("BornYear"));
            TempTeacher.SetAddress(RS.getString("Address"));
            TempTeacher.SetCity(RS.getString("City"));
            TempTeacher.SetCountry(RS.getString("Country"));
            TempTeacher.SetZipcode(RS.getString("Zipcode"));
            TempTeacher.SetOccupation(RS.getString("Occupation"));
            TempTeacher.SetPhone(RS.getString("Phone"));
            TempTeacher.SetEmail(RS.getString("Email"));
            TempTeacher.SetSQuestion(RS.getString("SQuestion"));
            TempTeacher.SetSAnswer(RS.getString("SAnswer"));
            Status = this.AddConnectedTeacher(TempTeacher);
          SQL = "Update Teachers SET Logedin = 1 WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "'";
            MyStatement.executeUpdate(SQL);
            Check = "Login";
          }
          else
          {
            Check = "WrongPassword";
       }
     }
     catch (SQLException sqle)
     {
         System.err.println(sqle.getMessage());
     }
     catch (ClassNotFoundException cnfe)
     {
         System.err.println(cnfe.getMessage());
     }
     catch (Exception e)
     {
       System.err.println(e.getMessage());
```

```
    }
    finally
    {
        try
        {
         if ( MyConn != null )
         {
          this.secretariat.ReleaseConnection(MyConn);
         }
        }
        catch (Exception e)
        {
         System.err.println(e.getMessage());
     }
    }
  }
  return (Status);
}

public synchronized int LoginAdministrator(User user)
{
  int Status = -1;
  String Check = new String("NotUsed");
  String CheckUser = new String();
  String CheckIdentify = new String();
  try
  {
   CheckUser = secretariat.CheckUser(user.GetUsername());
  }
  catch (Exception e)
  {
   System.err.println(e.getMessage());
  }
```

```
   CheckIdentify = secretariat.IdentifyUser(user.GetUsername());
   if ((CheckUser.equals("Used")) && (CheckIdentify.equals("Administrator")))
   {
    Connection MyConn = null;
    try
    {
        this.secretariat.LoadPool();
      MyConn = this.secretariat.GetConnection();
        Statement MyStatement = MyConn.createStatement();
        String SQL = "SELECT * FROM Administrators WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "' AND Logedin
= 0 ";
      ResultSet RS = MyStatement.executeQuery(SQL);
      if (RS.next())
        {
       VirtualAdministrator TempAdministrator = new VirtualAdministrator();
          TempAdministrator.SetAdministratorID(RS.getLong("AdministratorID"));
       TempAdministrator.SetIP(user.GetIP());
          TempAdministrator.SetUsername(RS.getString("Username"));
          TempAdministrator.SetPassword(RS.getString("Password"));
          TempAdministrator.SetName(RS.getString("Name"));
          TempAdministrator.SetLastname(RS.getString("Lastname"));
          TempAdministrator.SetSex(RS.getString("Sex"));
          TempAdministrator.SetDay(RS.getString("BornDay"));
          TempAdministrator.SetMonth(RS.getString("BornMonth"));
          TempAdministrator.SetYear(RS.getString("BornYear"));
          TempAdministrator.SetAddress(RS.getString("Address"));
          TempAdministrator.SetCity(RS.getString("City"));
          TempAdministrator.SetCountry(RS.getString("Country"));
          TempAdministrator.SetZipcode(RS.getString("Zipcode"));
          TempAdministrator.SetOccupation(RS.getString("Occupation"));
          TempAdministrator.SetPhone(RS.getString("Phone"));
          TempAdministrator.SetEmail(RS.getString("Email"));
```

```
            TempAdministrator.SetSQuestion(RS.getString("SQuestion"));
            TempAdministrator.SetSAnswer(RS.getString("SAnswer"));
            Status = this.AddConnectedAdministrator(TempAdministrator);
          SQL = "Update Administrators SET Logedin = 1 WHERE Username = '" +
user.GetUsername() + "' AND Password = '" + user.GetPassword() + "'";
            MyStatement.executeUpdate(SQL);
            Check = "Login";
          }
          else
          {
            Check = "WrongPassword";
          }
      }
      catch (SQLException sqle)
      {
        System.err.println(sqle.getMessage());
      }
      catch (ClassNotFoundException cnfe)
      {
          System.err.println(cnfe.getMessage());
      }
      catch (Exception e)
      {
          System.err.println(e.getMessage());
      }
      finally
      {
          try
        {
            if ( MyConn != null )
            {
              this.secretariat.ReleaseConnection(MyConn);
            }
```

```
      }
          catch (Exception e)
          {
           System.err.println(e.getMessage());
          }
     }
   }
   return (Status);
  }

  public synchronized String LogoutStudent(int Value)
  {
   String Check = new String();
   String Username = new String();
   VirtualStudent TempStudent = new VirtualStudent();
   TempStudent = this.GetConnectedStudent(Value);
   Username = TempStudent.GetUsername();
   Connection MyConn = null;
   try
   {
    this.secretariat.LoadPool();
    MyConn = this.secretariat.GetConnection();
    Statement MyStatement = MyConn.createStatement();
    String SQL = "Update Students SET Logedin = 0 WHERE Username = '" +
Username + "'";
    MyStatement.executeUpdate(SQL);
   }
   catch (SQLException sqle)
   {
    System.err.println(sqle.getMessage());
   }
   catch (ClassNotFoundException cnfe)
   {
```

```
    System.err.println(cnfe.getMessage());
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
   finally
   {
    try
    {
        if ( MyConn != null )
        {
       this.secretariat.ReleaseConnection(MyConn);
        }
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
   }
   this.RemoveConnectedStudent(Value);
   Check = "Logout";
   return (Check);
}

public synchronized void LogoutStudent(String Username)
{
   Connection MyConn = null;
   try
   {
    this.secretariat.LoadPool();
    MyConn = this.secretariat.GetConnection();
    Statement MyStatement = MyConn.createStatement();
```

```
    String SQL = "Update Students SET Logedin = 0 WHERE Username = '" +
Username + "'";
     MyStatement.executeUpdate(SQL);
   }
   catch (SQLException sqle)
   {
    System.err.println(sqle.getMessage());
   }
   catch (ClassNotFoundException cnfe)
   {
    System.err.println(cnfe.getMessage());
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
   finally
   {
    try
    {
        if ( MyConn != null )
        {
       this.secretariat.ReleaseConnection(MyConn);
        }
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
   }
   this.RemoveConnectedStudent(this.connectedStudents.IndexOf(Username));
  }
```

```
  public synchronized String LogoutTeacher(int Value)
  {
    String Check = new String();
    String Username = new String();
    VirtualTeacher TempTeacher = new VirtualTeacher();
    TempTeacher = this.GetConnectedTeacher(Value);
    Username = TempTeacher.GetUsername();
    Connection MyConn = null;
    try
    {
      this.secretariat.LoadPool();
      MyConn = this.secretariat.GetConnection();
      Statement MyStatement = MyConn.createStatement();
      String SQL = "Update Teachers SET Logedin = 0 WHERE Username = '" +
Username + "'";
      MyStatement.executeUpdate(SQL);
    }
    catch (SQLException sqle)
    {
      System.err.println(sqle.getMessage());
    }
    catch (ClassNotFoundException cnfe)
    {
      System.err.println(cnfe.getMessage());
    }
    catch (Exception e)
    {
      System.err.println(e.getMessage());
    }
    finally
    {
      try
      {
```

```
      if ( MyConn != null )
      {
        this.secretariat.ReleaseConnection(MyConn);
      }
    }
    catch (Exception e)
    {
      System.err.println(e.getMessage());
    }
  }
  this.RemoveConnectedTeacher(Value);
  Check = "Logout";
  return (Check);
 }

 public synchronized void LogoutTeacher(String Username)
 {
  Connection MyConn = null;
  try
  {
   this.secretariat.LoadPool();
   MyConn = this.secretariat.GetConnection();
   Statement MyStatement = MyConn.createStatement();
   String SQL = "Update Teachers SET Logedin = 0 WHERE Username = '" +
Username + "'";
   MyStatement.executeUpdate(SQL);
  }
  catch (SQLException sqle)
  {
   System.err.println(sqle.getMessage());
  }
  catch (ClassNotFoundException cnfe)
  {
```

```
    System.err.println(cnfe.getMessage());
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
   finally
   {
    try
    {
     if ( MyConn != null )
     {
      this.secretariat.ReleaseConnection(MyConn);
     }
    }
    catch (Exception e)
    {
     System.err.println(e.getMessage());
    }
   }
   this.RemoveConnectedTeacher(this.connectedTeachers.IndexOf(Username));
  }

  public synchronized String LogoutPrincipal()
  {
   String Check = new String();
   String Username = new String();
   Principal TempPrincipal = new Principal();
   TempPrincipal = this.GetConnectedPrincipal();
   Username = TempPrincipal.GetUsername();
   Connection MyConn = null;
   try
   {
```

```
    this.secretariat.LoadPool();
    MyConn = this.secretariat.GetConnection();
    Statement MyStatement = MyConn.createStatement();
    String SQL = "Update Teachers SET Logedin = 0 WHERE Username = '" +
Username + "'";
    MyStatement.executeUpdate(SQL);
   }
   catch (SQLException sqle)
   {
    System.err.println(sqle.getMessage());
   }
   catch (ClassNotFoundException cnfe)
   {
    System.err.println(cnfe.getMessage());
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
   finally
   {
    try
    {
     if ( MyConn != null )
         {
      this.secretariat.ReleaseConnection(MyConn);
         }
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
   }
```

```
   this.RemoveConnectedPrincipal();
   Check = "Logout";
   return (Check);
  }

  public synchronized String LogoutAdministrator(int Value)
  {
   String Check = new String();
   String Username = new String();
   VirtualAdministrator TempAdministrator = new VirtualAdministrator();
   TempAdministrator = this.GetConnectedAdministrator(Value);
   Username = TempAdministrator.GetUsername();
   Connection MyConn = null;
   try
   {
    this.secretariat.LoadPool();
    MyConn = this.secretariat.GetConnection();
    Statement MyStatement = MyConn.createStatement();
    String SQL = "Update Administrators SET Logedin = 0 WHERE Username = '" +
Username + "'";
    MyStatement.executeUpdate(SQL);
   }
   catch (SQLException sqle)
   {
    System.err.println(sqle.getMessage());
   }
   catch (ClassNotFoundException cnfe)
   {
    System.err.println(cnfe.getMessage());
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
```

```
    }
    finally
    {
     try
     {
      if ( MyConn != null )
      {
       this.secretariat.ReleaseConnection(MyConn);
      }
     }
     catch (Exception e)
     {
      System.err.println(e.getMessage());
     }
    }
    this.RemoveConnectedAdministrator(Value);
    Check = "Logout";
    return (Check);
  }

  public synchronized void LogoutAdministrator(String Username)
  {
    Connection MyConn = null;
    try
    {
     this.secretariat.LoadPool();
     MyConn = this.secretariat.GetConnection();
     Statement MyStatement = MyConn.createStatement();
     String SQL = "Update Administrators SET Logedin = 0 WHERE Username = '" +
Username + "'";
     MyStatement.executeUpdate(SQL);
    }
    catch (SQLException sqle)
```

```
  {
   System.err.println(sqle.getMessage());
  }
  catch (ClassNotFoundException cnfe)
  {
   System.err.println(cnfe.getMessage());
  }
  catch (Exception e)
  {
   System.err.println(e.getMessage());
  }
  finally
  {
   try
   {
    if ( MyConn != null )
    {
     this.secretariat.ReleaseConnection(MyConn);
    }
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
  }

this.RemoveConnectedAdministrator(this.connectedAdministrators.IndexOf(Usernam
e));
 }

 public synchronized void LogoutStudents()
 {
  Connection MyConn = null;
```

```
try
{
  this.secretariat.LoadPool();
  MyConn = this.secretariat.GetConnection();
  Statement MyStatement = MyConn.createStatement();
  String SQL = "Update Students SET Logedin = 0";
  MyStatement.executeUpdate(SQL);
}
catch (SQLException sqle)
{
  System.err.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
{
  System.err.println(cnfe.getMessage());
}
catch (Exception e)
{
  System.err.println(e.getMessage());
}
finally
{
  try
  {
       if ( MyConn != null )
       {
     this.secretariat.ReleaseConnection(MyConn);
       }
  }
  catch (Exception e)
  {
       System.err.println(e.getMessage());
  }
```

```
  }
  for (int i = 0; i < this.GetConnectedStudents().Size(); i++)
  {
   this.RemoveConnectedStudent(i);
  }
}

public synchronized void LogoutTeachers()
{
 Connection MyConn = null;
 try
 {
  this.secretariat.LoadPool();
  MyConn = this.secretariat.GetConnection();
  Statement MyStatement = MyConn.createStatement();
  String SQL = "Update Teachers SET Logedin = 0";
  MyStatement.executeUpdate(SQL);
 }
 catch (SQLException sqle)
 {
  System.err.println(sqle.getMessage());
 }
 catch (ClassNotFoundException cnfe)
 {
  System.err.println(cnfe.getMessage());
 }
 catch (Exception e)
 {
  System.err.println(e.getMessage());
 }
 finally
 {
  try
```

```
    {
        if ( MyConn != null )
        {
       this.secretariat.ReleaseConnection(MyConn);
        }
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
  }
  for (int i = 0; i < this.GetConnectedTeachers().Size(); i++)
  {
   this.RemoveConnectedTeacher(i);
  }
}

public synchronized void LogoutAdministrators()
{
  Connection MyConn = null;
  try
  {
   this.secretariat.LoadPool();
   MyConn = this.secretariat.GetConnection();
   Statement MyStatement = MyConn.createStatement();
   String SQL = "Update Administrators SET Logedin = 0";
   MyStatement.executeUpdate(SQL);
  }
  catch (SQLException sqle)
  {
   System.err.println(sqle.getMessage());
  }
  catch (ClassNotFoundException cnfe)
```

```
  {
   System.err.println(cnfe.getMessage());
  }
  catch (Exception e)
  {
   System.err.println(e.getMessage());
  }
  finally
  {
   try
   {
        if ( MyConn != null )
        {
      this.secretariat.ReleaseConnection(MyConn);
        }
   }
   catch (Exception e)
   {
        System.err.println(e.getMessage());
   }
  }
  for (int i = 0; i < this.GetConnectedAdministrators().Size(); i++)
  {
   this.RemoveConnectedAdministrator(i);
  }
}

public StudiesProgram LoadStudiesProgram(long Value) throws Exception
{
  String SQL;
  Connection MyConn = null;
  SixMonth TempSixMonth = new SixMonth();
  StudiesProgram TempStudiesProgram = new StudiesProgram();
```

```
try
{
 this.secretariat.LoadPool();
 MyConn = this.secretariat.GetConnection();
 Statement MyStatement = MyConn.createStatement();
 SQL = "SELECT StudiesProgram.StudiesProgramID, StudiesProgram.Name
FROM [StudiesProgram] WHERE ((([StudiesProgram].StudiesProgramID) = " +
Value + "))";
 ResultSet RSStudies = MyStatement.executeQuery(SQL);
 if (RSStudies.next())
 {
     TempStudiesProgram.SetID(RSStudies.getLong("StudiesProgramID"));
     TempStudiesProgram.SetName(RSStudies.getString("Name"));
     }
 RSStudies.close();
 SQL = "SELECT SixMonths.SixMonthID FROM [SixMonths] INNER JOIN
[SixMonthProgram] ON [SixMonths].SixMonthID =
[SixMonthProgram].ProgramSixMonthID ";
 SQL = SQL + "WHERE ((([SixMonthProgram].ProgramStudiesProgramID) = " +
Value + "))";
 ResultSet RS = MyStatement.executeQuery(SQL);
 while (RS.next())
 {
     try
     {
    TempSixMonth = this.LoadSixMonth(RS.getLong("SixMonthID"));
      TempStudiesProgram.SetSixMonth(TempSixMonth);
     }
     catch (Exception e)
     {
      System.err.println(e.getMessage());
     }
 }
```

```
    if (RS != null)
    {
        RS.close();
    }
  }
  catch (SQLException sqle)
  {
   System.err.println(sqle.getMessage());
  }
  catch (ClassNotFoundException cnfe)
  {
   System.err.println(cnfe.getMessage());
  }
  catch (Exception e)
  {
   System.err.println(e.getMessage());
  }
  finally
  {
   try
   {
    if ( MyConn != null )
    {
     this.secretariat.ReleaseConnection(MyConn);
    }
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
   }
  }
  return (TempStudiesProgram);
 }
```

```
 public SixMonth LoadSixMonth(long Value) throws Exception
 {
  String SQL;
  Connection MyConn = null;
  Lesson TempLesson = new Lesson();
  SixMonth TempSixMonth = new SixMonth();
  try
  {
   this.secretariat.LoadPool();
   MyConn = this.secretariat.GetConnection();
   Statement MyStatement = MyConn.createStatement();
   SQL = "SELECT SixMonths.SixMonthID, SixMonths.Name FROM [SixMonths]
WHERE ((([SixMonths].SixMonthID) = " + Value + "))";
   ResultSet RSSixMonth = MyStatement.executeQuery(SQL);
   if (RSSixMonth.next())
   {
       TempSixMonth.SetID(RSSixMonth.getLong("SixMonthID"));
       TempSixMonth.SetName(RSSixMonth.getString("Name"));
       }
   RSSixMonth.close();
   SQL = "SELECT Lessons.LessonID FROM [Lessons] INNER JOIN
[LessonProgram] ON [Lessons].LessonID = [LessonProgram].ProgramLessonID ";
   SQL = SQL + "WHERE ((([LessonProgram].ProgramSixMonthID) = " + Value +
"))";
   ResultSet RSLessons = MyStatement.executeQuery(SQL);
   while (RSLessons.next())
   {
       try
       {
         TempLesson = this.LoadLesson(RSLessons.getLong("LessonID"));
      TempSixMonth.SetLesson(TempLesson);
       }
```

```
        catch (Exception e)
        {
          System.out.println(e.getMessage());
    }
  }
  RSLessons.close();
}
catch (SQLException sqle)
{
  System.err.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
{
  System.err.println(cnfe.getMessage());
}
catch (Exception e)
{
  System.err.println(e.getMessage());
}
finally
{
  try
  {
        if ( MyConn != null )
        {
          this.secretariat.ReleaseConnection(MyConn);
        }
  }
  catch (Exception e)
  {
        System.err.println(e.getMessage());
  }
}
```

```
   return (TempSixMonth);
  }

  public Lesson LoadLesson(long Value) throws Exception
  {
   String SQL;
   Connection MyConn = null;
   Connection MyConn2 = null;
   Lesson TempLesson = new Lesson();
   try
   {
    this.secretariat.LoadPool();
    MyConn = this.secretariat.GetConnection();
    Statement MyStatement = MyConn.createStatement();
    SQL = "SELECT Lessons.LessonID, Lessons.LessonName, Section.SectionName
FROM [Section] INNER JOIN [Lessons] ON Section.SectionID =
Lessons.LessonSection WHERE ((([Lessons].LessonID) = " + Value + "))";
    ResultSet RS = MyStatement.executeQuery(SQL);
    if (RS.next())
    {
        TempLesson.SetID(RS.getLong("LessonID"));
        TempLesson.SetName(RS.getString("LessonName"));
        TempLesson.SetSection(RS.getString("SectionName"));
    }
    RS.close();
    SQL = "SELECT Teachers.TeacherID, Teachers.Name, Teachers.Lastname,
Teachers.Email FROM [Teachers] INNER JOIN [TeacherLessonProgram] ON
[Teachers].TeacherID = [TeacherLessonProgram].ProgramTeacher ";
    SQL = SQL + "WHERE ((([TeacherLessonProgram].ProgramLesson) = " + Value
+ "))";
    ResultSet RSTeacher = MyStatement.executeQuery(SQL);
    while (RSTeacher.next())
    {
```

```
            String CheckTeacher = new String();
            VirtualTeacher TempTeacher = new VirtualTeacher();
            TempTeacher.SetTeacherID(RSTeacher.getLong("TeacherID"));
            TempTeacher.SetName(RSTeacher.getString("Name"));
            TempTeacher.SetLastname(RSTeacher.getString("Lastname"));
            TempTeacher.SetEmail(RSTeacher.getString("Email"));
            TempLesson.SetTeacher(TempTeacher);
        }
        RSTeacher.close();
        try
        {
          MyConn2 = this.secretariat.GetConnection();
          Statement MyStatement3 = MyConn2.createStatement();
          String SQL3 = "SELECT LinkedLessonsLinkedLesson From LinkedLessons
Where (LinkedLessonsLessonID = " + Value + ")";
          ResultSet RS3 = MyStatement3.executeQuery(SQL3);
          while (RS3.next())
          {

TempLesson.AddRequiredLesson(RS3.getString("LinkedLessonsLinkedLesson"));
          }
          RS3.close();
        }
        catch (SQLException sqle)
        {
          System.err.println(sqle.getMessage());
        }
        catch (ClassNotFoundException cnfe)
        {
          System.err.println(cnfe.getMessage());
        }
        catch (Exception e)
        {
```

```
    System.err.println(e.getMessage());
  }
  finally
  {
       try
       {
        if ( MyConn2 != null )
        {
       this.secretariat.ReleaseConnection(MyConn2);
        }
       }
    catch (Exception e)
       {
        System.err.println(e.getMessage());
       }
  }
}
catch (SQLException sqle)
{
  System.err.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
{
  System.err.println(cnfe.getMessage());
}
catch (Exception e)
{
  System.err.println(e.getMessage());
}
finally
{
  try
  {
```

```
            if ( MyConn != null )
            {
              this.secretariat.ReleaseConnection(MyConn);
            }
        }
        catch (Exception e)
        {
            System.err.println(e.getMessage());
        }
      }
      return (TempLesson);
    }
}
```