ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΠΜΣ στην Επιστήμη και Τεχνολογία Υπολογιστών

Διπλωματική εργασία

Καλδή Χριστίνα

Θέμα: "Δημιουργία ηλεκτρονικής εφαρμογής για την οργάνωση και διαχείριση ακαδημαϊκής τάξης - MyClass"

Επιβλέποντες καθηγητές : Μασσέλος Κωνσταντίνος
Δημητρουλάκος Γρηγόριος

Σεπτέμβριος 2016

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία έχει ως κύριο στόχο την ανάπτυξη μιας ηλεκτρονικής εφαρμογής σε περιβάλλον Microsoft Visual Studio και συγκεκριμένα Windows Form μορφής με χρήση γλώσσας προγραμματισμού C#. Η εφαρμογή επιτρέπει στον χρήστη τόσο την νέα καταχώρηση στοιχείων για φοιτητές και καθηγητές όσο και την αναζήτηση ήδη καταχωρημένων εγγραφών.

Αναλυτικότερα η εφαρμογή επιτρέπει στον χρήστη να:

- καταχωρήσει ή αναζητήσει φοιτητές , και συγκεκριμένα στοιχεία που αφορούν προσωπικά δεδομένα, μαθήματα παρακολούθησης, βαθμολογία μαθημάτων και απουσίες,
- καταχωρήσει ή αναζητήσει καθηγητές, και συγκεκριμένα στοιχεία που αφορούν προσωπικά δεδομένα και αναθέσεις μαθημάτων,
- επιλέξει μαθήματα για φοιτητές και καθηγητές απο συγκεκριμένο αρχείο,
- δημιουργήσει τμήματα δικής του επιλογής καθώς και να ορίσει κριτήρια βαθμολόγησης του κάθε μαθήματος,
- αποθηκεύσει όλες τις καταχωρήσεις σε xml αρχείο,
- εμφανίσει σε αρχείο excel για κάθε μάθημα και τμήμα στοιχεία όσον αφορά στη βαθμολογία, στις απουσιές και στο τελικό αποτέλεσμα.

Αν θελήσουμε να συνοψίσουμε, τα βασικά στοιχεία αυτής της διπλωματικής εργασίας επικεντρώνονται τόσο στη λειτουργικότητα και ευχριστία της ηλεκτρονικής εγαρμογής σε περιβάλλον Visual Studio, όσο και στην ευκολία επέκτασης της εφαρμογής για περαιτέρω απαιτήσεις εξειδικευμένων χρηστών.

Συμπερασματικά, πρόκειται για μια πλήρη ηλεκτρονική εφαρμογή οργάνωσης και διαχείρισης ηλεκτρονικής τάξης που μπορεί να χρησιμοποιηθεί από καθηγητές χωρίς ιδιαίτερες γνώσεις για το περιβάλλον ανάπτυξης. Παράλληλα όμως η εφαρμογή διαμορφώνεται εύκολα και είναι έτοιμη να ακολουθήσει τυχόν μελλοντικές αναγκές χρηστών με εξειδικευμένες γνώσεις σε αυτό το προγραμματιστικό περιβάλλον και της γλώσσας προγραμματισμού C#.

Abstract

The main goal of this diploma thesis is the development of an electronic application within the Microsoft Visual Studio environment and specific in Windows Form by using C# programming language. The application allows the user to both the new data entry for students and teachers and the search already posted entries.

Specifically, the application allows the user to:

- register or search for students, namely information relating to personal data, registered lessons, score and absences,
- register or search for teachers, namely information relating to personal data and lessons assignments,
- select courses for students and teachers from specified file,
- create classes of his choice and to set criteria for scoring of each lesson,
- save all entries to a .xml file,
- display in excel file for each course and class information regarding the score, the abscences and the final result.

To sum up, the basic elements of this diploma thesis focus both in functionality and usability of the electronic application into Visual Studio environment, and the ease of extending the scope for further requirements of specialized users.

In conclusion, this is a complete electronic application for organizing and managing a class that can be used by teachers without special knowledge for the development environment. However the application is formed easily and is ready to follow any future requirements of users with specialized knowledge in this programming environment and programming language C#.

ΠΕΡΙΕΧΟΜΕΝΑ

1

Εισαγωγή

1.1 Αντικείμενο της διπλωματικής

Σήμερα, περισσότερο από ποτέ, οι ηλεκτρονικές και διαδικτυακές εφαρμογές δίνουν τη δυνατότητα στους χρήστες να ανταποκριθούν στις αυξημένες απαιτήσεις της εποχής για οποιοδήποτε τομέα τους αφορά. Τόσο η ανάπτυξη όσο και η διαχείριση των εφαρμογών γίνεται μέσα από ειδικές πλατφόρμες που προσφέρουν ολοκληρωμένες λύσεις, μια τέτοια είναι και το Visual Studio της Microsoft που παρέχει πλήρη και απλοποιημένα εργαλεία για μοντέρνες και έξυπνες εφαρμογές. Το Visual Studio κυκλοφορεί σε πολλές εκδόσεις και υποστηρίζει διάφορες γλώσσες προγραμματισμού όπως τη Visual Basic, C#, C, C++ κ.α. Περιλαμβάνει έναν επεξεργαστή κώδικα (code editor) με αυτόματη συμπλήρωση (IntelliSence), που επιταχύνει το γράψιμο και ενσωματώνει έναν debugger (αποσφαλματωτή) για τον άμεσο εντοπισμό σφαλμάτων. Η δική μας εφαρμογή έχει αναπτυχθεί με την έκδοση Visual Studio 2013, σε γλώσσα προγραμματισμού C# και είναι σε παραθυρική μορφή (WindowsForm Application).

Σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη μιας εφαρμογής που δίνει τη δυνατότητα στο χρήστη να οργανώσει και να διαχειριστεί μια ηλεκτρονική τάξη. Συγκεκριμένα, μπορεί να εισάγει στοιχεία για καθηγητές και φοιτητές/μαθητές, να δημιουργεί τμήματα στα αντίστοιχα μαθήματα παρακολούθησης, να ορίζει τα κριτήρια βαθμολόγησης και τη βαρύτητά τους. Στη συνέχεια μπορεί να καταχωρεί βαθμολογίες και απουσίες και να αντλεί πληροφορίες που αφορούν τους βαθμούς, τις απουσίες και το τελικό αποτέλεσμα επίδοσής τους. Τέλος, όλα τα στοιχεία που καταχωρούνται βρίσκονται αποθηκευμένα σε ένα .xml αρχείο και οι πληροφορίες που αντλούμε υπάρχει η δυνατότητα να εμφανίζονται και σε αρχείο excel και να δέχονται περαιτέρω επεξεργασία, ανάλογα με τις απαιτήσεις του χρήστη.

1.2 Οργάνωση Κειμένου

Η εργασία αυτή αναλύεται σε 5 συνολικά κεφάλαια και περιγράφουμε συνοπτικά τι πραγματεύεται το καθένα από αυτά:

Κεφάλαιο 1ο: γνωρίζουμε στον αναγνώστη το περιβάλλον το οποίο έχει χρησιμοποιηθεί για την παρούσα διπλωματική εργασία και τις δυνατότητές του. Στη συνέχεια, παρουσιάζουμε σύντομα το σκοπό και τη χρησιμότητα της εφαρμογής μας.

Κεφάλαιο 2ο: ασχολείται αναλυτικότερα με το περιβάλλον ανάπτυξης της εφαρμογής, βλέπουμε διάφορες εκδόσεις και τον τρόπο εγκατάστασής του. Έπειτα γνωρίζουμε τη γλώσσα προγραμματισμού C#, τα βασικά της χαρακτηριστικά και τις δυνατότητες που μας προσφέρει. Αναπτύσσουμε μια απλή εφαρμογή σε WindowsForm μορφή για να δούμε ενδεικτικά λίγα από τα βασικά εργαλεία και λειτουργίες που έχουν χρησιμοποιηθεί στην εφαρμογή μας. Τέλος παρουσιάζεται ο τρόπος που δημιουργούμε ερωτήματα με τη γλώσσα Linq, αφού μας είναι απαραίτητα για να αντλήσουμε τις κατάλληλες πληροφορίες από τα καταχωρημένα στοιχεία μας.

Κεφάλαιο 3ο: σε αυτό το κεφάλαιο παρουσιάζονται γενικές πληροφορίες για κάποιους τύπους αρχείων που χρησιμοποιήθηκαν στην εφαρμογή μας. Περιγράφονται τα .xml αρχεία, τέτοιου τύπου είναι το αρχείο που αποθηκεύονται όλα τα δεδομένα που καταχωρούμε κατα την χρήση της εφαρμογής. Καθώς και τα αρχεία .csv που είναι η μορφή που μετατρέπουμε τα δεδομένα μας πρίν δημιουργηθεί το αρχείο excel.

Κεφάλαιο 4ο: εδώ παρουσιάζονται αναλυτικά όλες οι λειτουργίες της εφαρμογής και οι επιλογές που δίνονται στον χρήστη κατα τη λειτουργία της. Περιγράφεται ξεχωριστά η κάθε καρτέλα της εφαρμογής καθώς και οι επιμέρους οθόνες της κάθε καρτέλας ώστε ο αναγνώστης να γνωρίζει πλήρως τις δυνατότητες που του προσφέρονται. Εκτός απο τις εικόνες που εμφανίζονται, υπάρχουν και τμήματα κώδικα που ίσως βοηθούν στην καλύτερη κατανόηση κάποιων σημαντικών λειτουργιών της εφαρμογής μας.

Κεφάλαιο 5ο: αναφέρονται τα συμπεράσματα που προκύπτουν από την εκπόνηση αυτής της εργασίας και μελλοντικές επεκτάσεις της εφαρμογής για επιπλέον λειτουργίες.

Στο *ΠΑΡΑΡΤΗΜΑ* της εργασίας υπάρχει αναλυτικά ο κώδικας της εφαρμογής που αναπτύχθηκε, οργανωμένος σε κλάσεις των δυο project που αποτελούν την εφαρμογή.

Τέλος παρουσιάζεται η *ΒΙΒΛΙΟΓΡΑΦΙΑ* της παρούσας διπλωματικής εργασίας και όλες οι ιστοσελίδες που βοήθησαν στην ολοκλήρωση της εργασίας αλλά και της εφαρμογής.

2

Προγραμματισμός σε περιβάλλον Visual Studio

2.1 Το Visual Studio

Το Visual Studio είναι το Integrated Development Environment της Microsoft για την ανάπτυξη εφαρμογών που στοχεύουν στο .NET Framework και άλλες πλατφόρμες της Microsoft: από εφαρμογές κονσόλας (console applications) μέχρι web services και εγγενή κώδικα (native code).

Το Visual Studio περιλαμβάνει πλήθος εργαλείων για την υποστήριξη της ανάπτυξης εφαρμογών από τα πρώτα στάδια μέχρι και τα τελικά. Περιλαμβάνει έναν επεξεργαστή κώδικα (code editor) με αυτόματη συμπλήρωση (IntelliSence), που επιταχύνει το γράψιμο και ενσωματώνει έναν debugger (αποσφαλματωτή) για τον άμεσο εντοπισμό σφαλμάτων. Άλλα εργαλεία περιλαμβάνουν την σχεδίαση παραθυρικών εφαρμογών και τη σχεδίαση κλάσεων και βάσεων δεδομένων, ενώ μπορεί να επεκταθεί με τα διάφορα επιπρόσθετα εργαλεία που κυκλοφορούν από τη Microsoft ή τρίτους παρόχους.

Για φοιτητές τα εργαλεία του Visual Studio προσφέρονται δωρεάν μέσα από το πρόγραμμα Dreamspark (www.dreamspark.com). Γενικότερα, πολλές εκδόσεις του Visual Studio προσφέρονται δωρεάν σε επίσημες ιστοσελίδες όπως https://www.visualstudio.com και https://www.microsoft.com.

Το Visual Studio έχει ένα πακέτο από προεγκατεστημένες γλώσσες προγραμματισμού με την εγκατάστασή του,αυτές είναι η C,C++,C#,Visual Basic,F#,SQL και Javascript. Ωστόσο , μπορούμε να προσθέσουμε την Python και την PHP.

Εκδόσεις του Visual Studio

Υπάρχουν πολλαπλές εκδόσεις του Visual Studio

- **Visual Studio Express**

Οι εκδόσεις Visual Studio Express είναι ένα σύνολο ελαφρών ανεξάρτητων IDEs που παρέχονται σαν απογυμνωμένες εκδόσεις του Visual Studio δωρεάν, ανά πλατφόρμα και ανά γλώσσα. Πιο συγκεκριμένα οι εκδόσεις περιλαμβάνουν τα εργαλεία ανάπτυξης για τις αντίστοιχες πλατφόρμες (web, windows, phone) ή για τις αντίστοιχες γλώσσες (VB, C#). Το Visual Studio Express περιλαμβάνει μόνο ένα υποσύνολο των εργαλείων που παρέχονται στις άλλες εκδόσεις. Επίσης δεν υποστηρίζει plug-ins. Οι μεταγλωττιστές για 64-bit αρχιτεκτονικές δεν συμπεριλαμβάνονται στις εκδόσεις express, ωστόσο μπορούν να εγκατασταθούν ξεχωριστά. Οι εκδόσεις express στοχεύουν στην εκπαίδευση και στους χομπίστες. Τέλος οι express εκδόσεις δεν χρησιμοποιούν την πλήρη MSDN Library αλλά την MSDN Essentials Library.

- **Visual Studio LightSwitch**

Το Visual Studio LightSwitch είναι ένα απλοποιημένο IDE που επιτρέπει τη δημιουργία εταιρικών εφαρμογών γρήγορα και εύκολα για το desktop ή το cloud. Οι εφαρμογές που παράγονται είναι αρχιτεκτονικά 3 επιπέδων: η διεπαφή χρήστη τρέχει σε Microsoft Silverlight,η λογική και το επίπεδο πρόσβασης δεδομένων βασίζεται σε WCF RIA Services και το Entity Framework ενώ φιλοξενείται στην ASP.NET. Επίσης υποστηρίζει και τον SQL Server αλλά και το Microsoft SQL Azure για την αποθήκευση δεδομένων. Περιλαμβάνει περιβάλλοντα σχεδιάσης για το σχεδιασμό οντοτήτων και συσχετίσεων αλλά και γραφικών διεπαφών. Η λογική της εφαρμογής μπορεί να γραφτεί σε Visual Basic ή C#. η έκδοση αυτή μπορεί να εγκατασταθεί αυτόματα ή μέσα στο Visual Studio Professional.

- **Visual Studio Professional**

Το Visual Studio Professional παρέχει ένα IDE για όλες τις υποστηριζόμενες γλώσσες στο .NET. Απλουστεύει τη δημιουργία, την αποσφαλμάτωση και την εγκατάσταση εφαρμογών σε πολλές πλατφόρμες, που συμπεριλαμβάνουν το SharePoint και το Cloud. Η υποστήριξη MSDN διατίθεται στην έκδοση Essentials ή στην πλήρη έκδοση, ανάλογα με την άδεια που αγοράστηκε. Υποστηρίζει την επεξεργασία XML και XSLT και μπορεί να δημιουργήσει πακέτα εγκατάστασης που χρησιμοποιούν τις τεχνολογίες ClickOnce και MSI. Συμπεριλαμβάνει εργαλεία όπως τον Server Explorer για την ενοποίηση με τον SQL Server.

- **Visual Studio Premium**

Η έκδοση Premium είναι ένα πλήρες πακέτο εργαλείων που απλουστεύει την ανάπτυξη εφαρμογών ατομικά αλλά και στα πλαίσια ομάδων ανάπτυξης. Ουσιαστικά, περιλαμβάνει όλα τα εργαλεία της έκδοσης Professional, αλλά και εργαλεία για την ανάλυση του κώδικα, εργαλεία για profiling αλλά και για database unit testing.

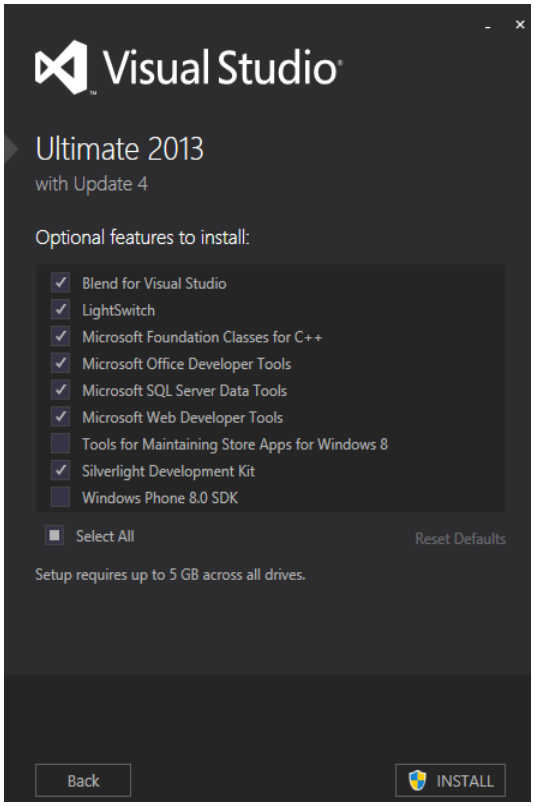
- **Visual Studio Ultimate**

Η έκδοση Ultimate αποτελεί μετονομασία της έκδοσης Team Suite. Η έκδοση αυτή παρέχει ένα σύνολο εργαλείων για την υποστήριξη της συνεργατικής ανάπτυξης έργων στο .NET. Επιπρόσθετα των χαρακτηριστικών του Visual Studio Premium, συμπεριλαμβάνονται και εργαλεία για την ανάπτυξη βάσεων δεδομένων, την ανάλυση του κώδικα, την αρχιτεκτονική μιας εφαρμογής, τη δοκιμή αλλά και εργαλεία reporting.

Εγκατάσταση

Για να εγκαταστήσουμε το Visual Studio 2013 Ultimate στον υπολογιστή μας ακολουθούμε τα επόμενα βήματα:

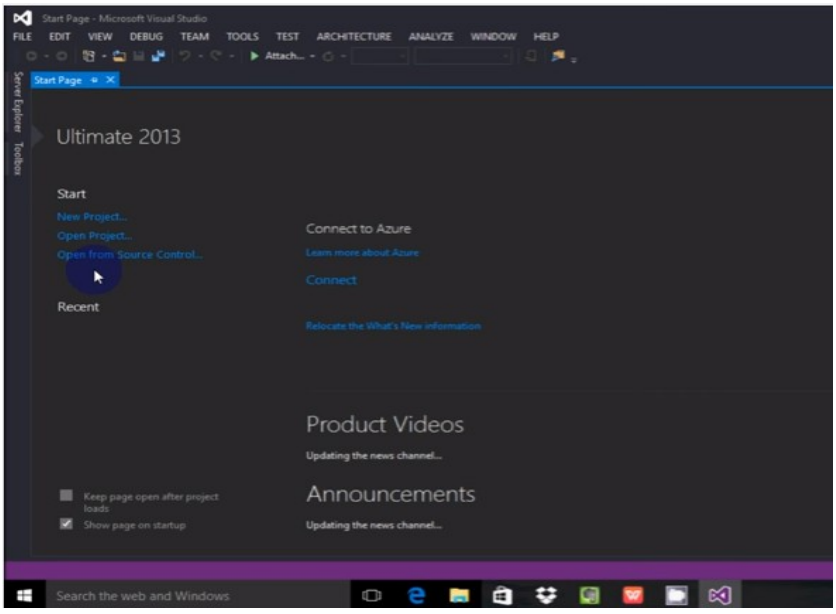
- Έχοντας πρόσβαση στο Web, από το website της Microsoft κατεβάζουμε τον οδηγό εγκατάστασης για το Visual 2013 Ultimate (vs_ultimate.exe) και τρέχουμε το αρχείο.
- Επιλέγουμε κάποια προαιρετικά εργαλεία, αν είναι απαραίτητα και προχωρούμε στην εγκατάσταση (Install).



- Η διαδικασία εγκατάστασης αρχίζει και πρέπει να περιμένουμε κάποια λεπτά μέχρι να ολοκληρωθεί ή διαφορετικά να πατήσουμε το πλήκτρο Cancel για να ακυρωθεί.

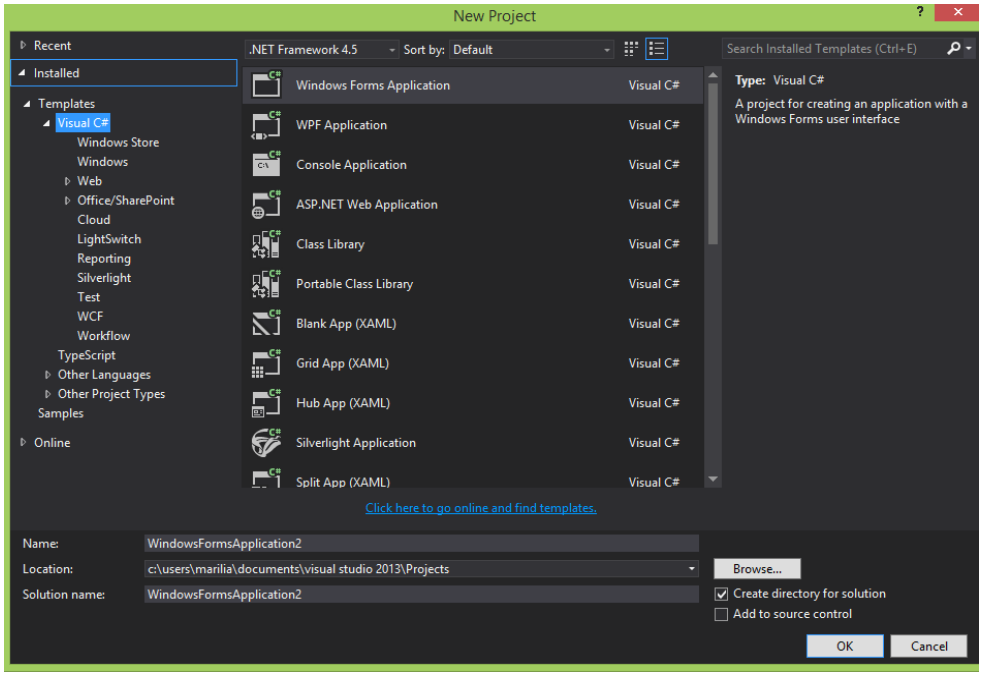
- Όταν ολοκληρωθεί η εγκατάσταση επιτυχώς, θα ζητηθεί να γίνει επαννεκίνηση του υπολογιστή επιλέγοντας Restart Now.

- Μετα την επαννεκίνηση το Visual Studio είναι έτοιμο για χρήση και η αρχική οθόνη μοιάζει κάπως έτσι..



- Επιλέγοντας New Project μπορούμε να δημιουργήσουμε το πρώτο μας project
- Από τα Templates επιλέγουμε Visual C# για να χρησιμοποιήσουμε τη C#, και με

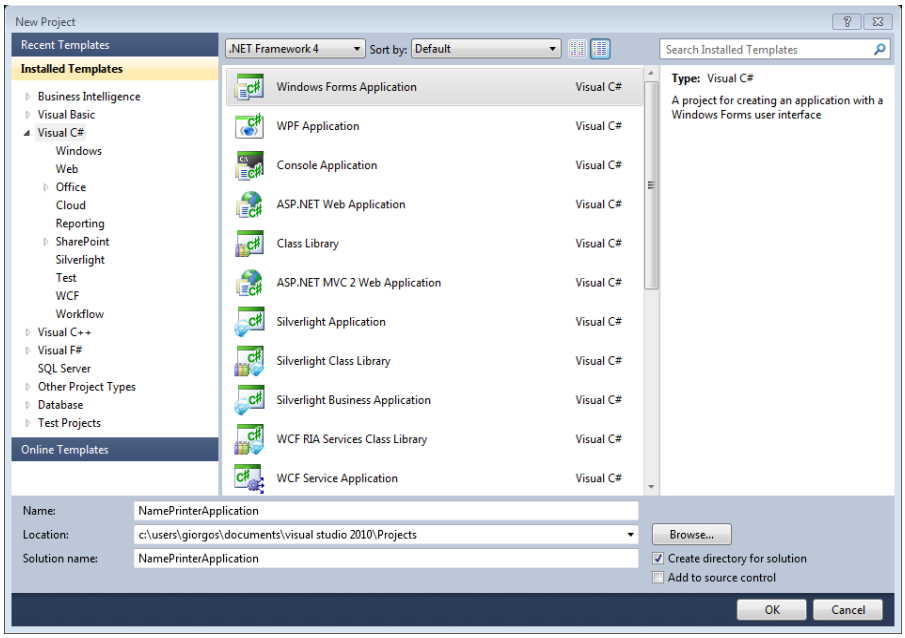
την επιλογή WindowsForm Application δημιουργούμε την πρώτη μας εφαρμογή παραθυρικής μορφής.



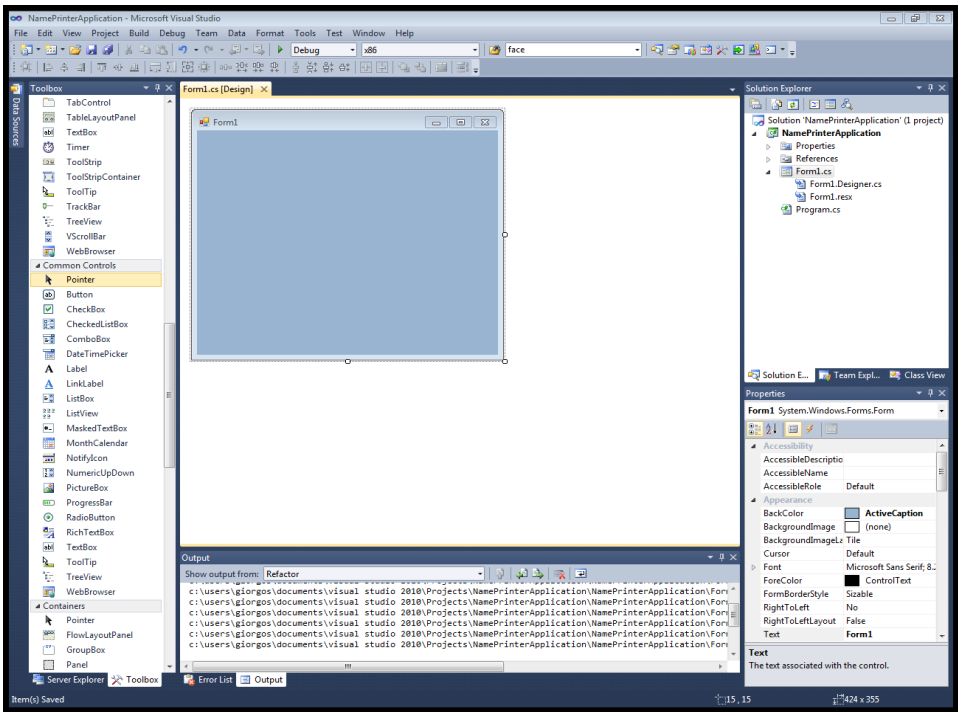
2.2 Δημιουργία μιας απλής Windows Form

Ακολουθεί η δημιουργία μιας μικρής εφαρμογής όπου ένας χρήστης θα γράφει το όνομα του, το επίθετο του και αυτά θα εμφανίζονται είτε σε ένα τρίτο TextBox είτε σε ένα MessageBox. Η δημιουργία αυτή θα είναι ένα απλό παραδείγμα Windows Form, ως ένα μικρό δείγμα λίγων εργαλείων από όσα θα χρησιμοποιηθούν στην εφαρμογή που ακολουθεί στην παρούσα διπλωματική εργασία.

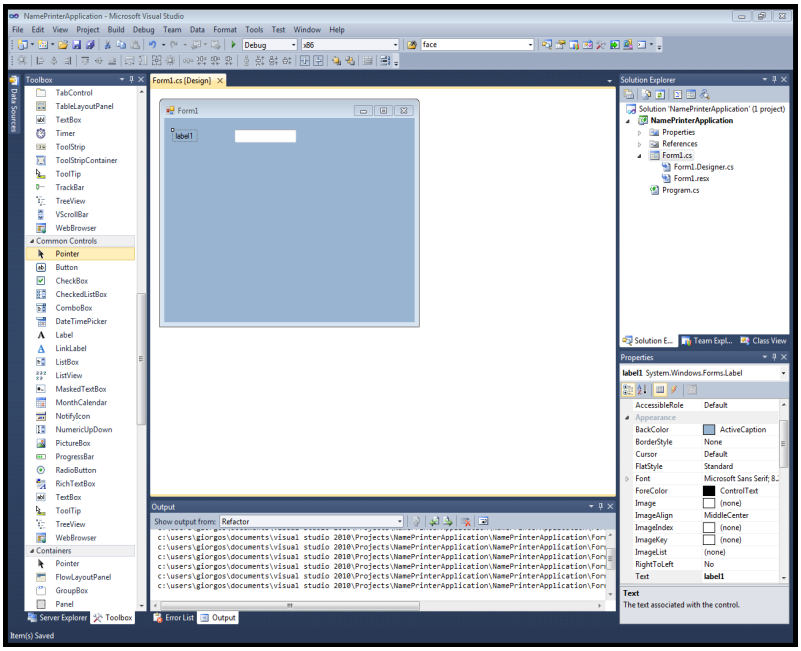
Ανοίγοντας το Visual Studio, δίνεται η επιλογή New Project και στα αριστερά Visual C#.Στο κεντρικό παράθυρο δίνεται η επιλογή 'Windows Form Application' όπως και στην παρακάτω εικόνα.



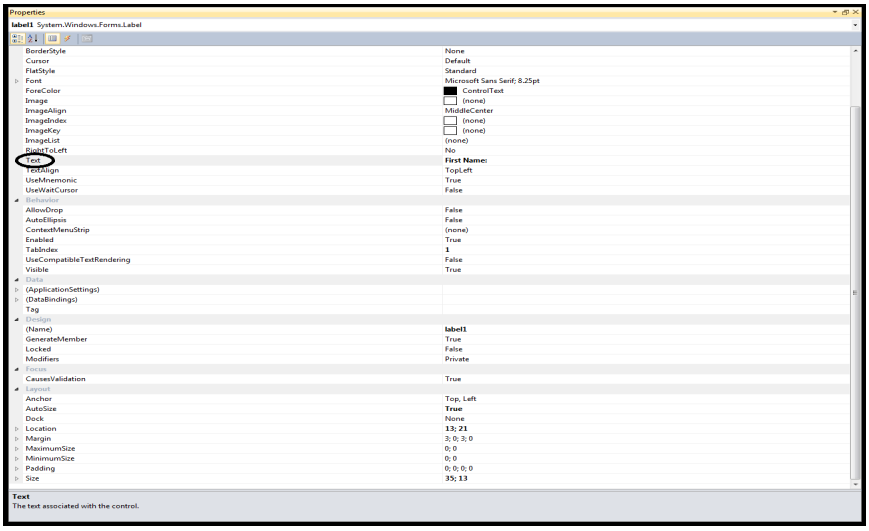
Δίνεται κατά προτίμηση ένα name,στη συγκεκριμένη περίπτωση δόθηκε το παραπάνω,δηλαδή 'NamePrinterApplication' και OK. Γίνεται άνοιγμα του Design window του Visual Studio και εμφανίζεται η κενή φόρμα. Υπάρχει και η επιλογή αλλαγής του χρώματος.



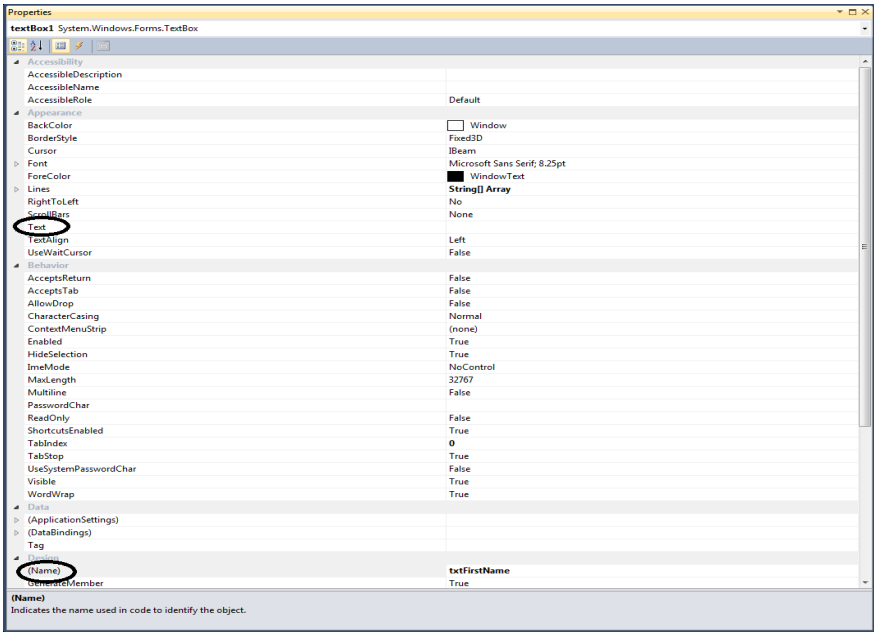
Στο Toolbox αριστερά βρίσκεται το control 'Label' (όλα τα αντικείμενα που βρίσκονται στο Toolbox, όπως Buttons, Labels κτλ ονομάζονται controls) και σέρνοντάς το στην Form, στο κεντρικό παράθυρο και στην συνέχεια πάλι στο Toolbox, εντοπίζοντας το 'Textbox' και σέρνοντάς το και αυτό στην Form πρέπει να έχει την παρακάτω μορφή:



Πατώντας κλικ πάνω στο Label που έχει προστεθεί (το label1 όπως φαίνεται και στην φόρμα).Κάτω δεξιά υπάρχει μια περιοχή που λέγετε Properties (αν δεν είναι εμφανής,κάνοντας δεξί κλικ στο Label και επιλέγοντας Properties, θα γίνει ορατή).Εκεί βρίσκεται το Text property όπου γίνεται αλλαγή στο οποιοδήποτε κείμενο μπορεί να έχει στο 'First Name: ', όπως παρακάτω:



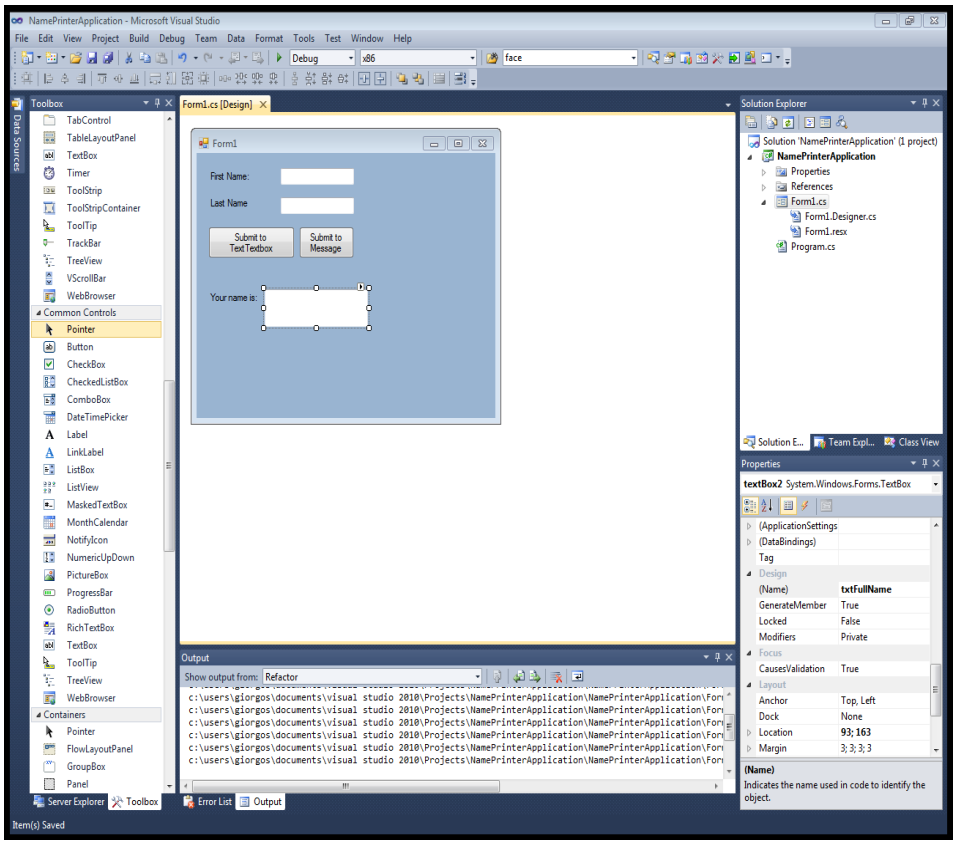
Τώρα πρέπει να γίνει εισαγωγή και ενός TextBox. Βρίσκεται στο Toolbox αριστερά, εκεί που βρίσκεται και το Label. Σέρνοντάς το στην φόρμα κάνουμε κλικ πάνω του.Πηγαίνοντας στα Properties του και εντοπίζοντας το Text property (όπως και πριν),κάνουμε διαγραφή σε ότι κείμενο μπορεί να έχει. Εντοπίζοντας και το Name property(βρίσκεται λίγο πιο κάτω)γίνεται αλλαγή σε 'txtFirstName'. Ενδεικτικά:



Προσθέτουμε άλλα 2 Label, 2 TextBoxes και 2 Buttons στην φόρμα και ακολουθώντας τα παρακάτω βήματα:

Label --> Text = Last Name:
Label --> Text = Your name is:
TextBox --> Text = '' (δηλαδή κενό) , Name = 'txtLastName'
TextBox --> Text = '' (δηλαδή κενό) , Name = 'txtFullName', MultiLine = True
Button --> Text = 'Submit to Textbox', Name = 'btnToText'
Button --> Text = 'Submit to Message', Name = 'btnToMessage'

Η φόρμα θα πρέπει να μοιάζει με την παρακάτω:



Γίνεται αλλαγή του Name property των controls γιατί αυτό το Name θα χρειαστεί στον κώδικα. Σ' ένα TextBox έγινε ορισμός ενός νέου Property, το 'MultiLine = True'. Με αυτό δίνετε η εντολή στο .NET να κάνει αυτό το TextBox πολλών γραμμών, δηλαδή σαν ένα TextArea(από προεπιλογή είναι μόνο μιας γραμμής),για πλήρη κάλυψη σε περίπτωση που το όνομα που θα εισάγουμε είναι αρκετά μεγάλο.Έτσι γίνεται η δημιουργία μιας Windows Form εφαρμογής! Πατώντας στην Form δεξί κλικ και επιλέγοντας 'View code' εμφανίζετε ο κώδικας που έχει δημιουργηθεί.

```
namespace NamePrinterApplication
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Η δουλειά της μεθόδου InitializeComponent είναι να αρχικοποιεί όλα τα controls(textboxes, buttons κτλ). Κάθε φορά που θα πατιέται το 'Submit to Textbox' θα παίρνει ότι έχει γίνει εισαγωγή στα παραπάνω TextBoxes και θα εμφανίζει το όνομα στο TextBox που βρίσκεται κάτω(στο txtFullName). Για να γίνει αυτό πρέπει στο Design να πατηθεί διπλό κλικ στο Button.

Μέσα στην μέθοδο έχει γραφτεί ο κώδικας για να αποσπαστεί το κείμενο από τα

```
//Μεθοδος που καλειται καθε φορα που γινεται κλικ στο Button 'Submit to TextBox'
private void btnToText_Click(object sender, EventArgs e)
{
    //Το fullname textbox αρχικοποιειται σε κενο καθε φορα που το παταμε για να το κ
    txtFullName.Text = "";

    //Παιρνουμε το περιεχομενο του πρωτου TextBox και το βαζουμε σε μια string μεταβ
    string firstName = txtFirstName.Text;

    //Παιρνουμε το περιεχομενο του δευτερου TextBox και το βαζουμε σε μια string μετ
    string lastName = txtLastName.Text;

    //Αν το string firstName ΔΕΝ ειναι ισο με το κενο string
    //και το string lastName ΔΕΝ ειναι ισο με το κενο string
    if (!(firstName.Equals("")) && !(lastName.Equals("")))
    {
        //Το τριτο TextBox παιρνει την τιμη του firstName και του lastName
        txtFullName.Text = firstName + " " + lastName;
    }//τελος if
    else //αν καποιο απο αυτα τα string ηταν κενο δηλαδη ο χρηστης δεν εδωσε ονομα
    {
        //εμφανιζει ενα μυνημα στο TextBox
        txtFullName.Text = "No name entered";
    }//τελος else
}//τελος μεθοδου
```

textboxes. Χρησιμοποιήθηκε το property Text για να αποσπαστεί το κείμενο από το κάθε TextBox. Για την αναγνώριση του κάθε TextBox δίνεται ένα μοναδικό Name (αυτή η ενέργεια έχει γίνει προηγουμένως, όταν δημιουργήθηκαν τα controls). Το πρώτο TextBox είναι το txtFirstName (για να γίνει η αναφορά σε αυτό μέσω του κώδικα), το δεύτερο είναι το txtLastName και το txtFullName είναι αυτό που θα πάρει το αποτέλεσμα την σύμπτυξης των δύο strings. Για τον έλεγχο αν ένα string ισούται με κάτι χρησιμοποιείται η μέθοδος Equals,όπως παραπάνω,περνώντας ως όρισμα της το string που χρειάζεται για να,ελεχθεί για ισότητα.Με τον τελεστή '!' μπροστά ορίζεται η άρνηση, δηλαδή στο παραπάνω,αν ΔΕΝ είναι ίσο με ""(κενό string). Αυτά για το πρώτο button. Για το δεύτερο ο κώδικας αναμένεται περίπου ο ίδιος. Στο Design ξανά,γίνεται διπλό κλικ στο δεύτερο Button για τη δημιουγία του.

```
//Μεθοδος που καλειται καθε φορα που γινεται κλικ στο Button 'Submit to Mess
        private void btnToMessage_Click(object sender, EventArgs e)
        {
            //Παιρνουμε το περιεχομενο του πρωτου TextBox και το βαζουμε σε μι
            string firstName = txtFirstName.Text;

            //Παιρνουμε το περιεχομενο του δευτερου TextBox και το βαζουμε σε
            string lastName = txtLastName.Text;

            //Αν το string firstName ΔΕΝ ειναι ισο με το κενο string
            //και το string lastName ΔΕΝ ειναι ισο με το κενο string
            if (!(firstName.Equals("")) && !(lastName.Equals("")))
            {
                //Εμφανιζεται ενα alert μυνημα με το ονομα και το επιθετο που
                MessageBox.Show(firstName + " " + lastName);
            }//τελος if
            else //αν καποιο απο αυτα τα string ηταν κενο δηλαδη ο χρηστης δεν
            {
                //εμφανιζει ενα μυνημα
                MessageBox.Show("No name entered");
            }//τελος else
        }//τελος μεθοδου
```

Ο κώδικας είναι ίδιος, η μόνη διαφορά είναι πως τώρα εμφανίζονται τα δεδομένα σε ένα Alert Message box, παρά στο textbox. Αυτό επιτυγχάνεται αν γραφτεί MessageBox.Show("Κείμενο εδώ"). Δείχνει το MessageBox στην οθόνη μαζί με το περιεχόμενο που δόθηκε ως παράμετρος(που πρέπει να είναι string).

2.3 Η γλώσσα προγραμματισμού C#

Η C# είναι μια γλώσσα απλή, μοντέρνα, αντικειμενοστρεφής και έχει στοιχεία από τις C, C++ και JAVA γλώσσες προγραμματισμού. Συντακτικά η C# μοιάζει πολύ στη C++ και στη JAVA καθώς πολλές λέξεις-κλειδιά είναι ίδιες. Επίσης μοιράζεται με αυτές,

α) τη δομή των μπλοκ τα οποία ορίζονται με τα σύμβολα "{" και "}" αλλά και

β) την οριοθέτηση των εντολών με το σύμβολο ";".

Μπορεί με την πρώτη ματιά ενός κομματιού κώδικα σε C# να πούμε ότι είναι εμφανής η ομοιότητα με τις γλώσσες C++ και JAVA αλλά ή αλήθεια είναι ότι η C# είναι πιο εύκολη στην εκμάθηση από την πρώτη και παρόμοιας δυσκολίας με την δεύτερη. Ο σχεδιασμός της είναι συνυφασμένος με τα μοντέρνα προγραμματιστικά πρότυπα και παράλληλα έχει υιοθετήσει την απλότητα χρήσης της Visual Basic αλλά και την υψηλή απόδοση της C++ σε θέματα χαμηλού επιπέδου διαχείρισης της μνήμης, σε περίπτωση που χρειαστεί.

Τα κύρια χαρακτηριστικά της C# είναι τα εξής:

- Πλήρης υποστήριξη για κλάσεις και αντικειμενοστρεφή προγραμματισμό, η οποία περιέχει κληρονομικότητα διεπαφής και υλοποίησης, εικονικές συναρτήσεις και υπερφόρτωση τελεστών.

- Ένα συνεπές και καθορισμένο με σαφήνεια σύνολο βασικών τύπων μεταβλητών.

- Δεν υπάρχουν καθολικές μεταβλητές και μέθοδοι. Όλες οι μέθοδοι πρέπει να δηλώνοναι μέσα στις κλάσεις. Οι δημόσιες (public) κλάσεις μπορούν να αντικαταστήσουν τις καθολικές μεταβλητές και τις καθολικές συναρτήσεις.

- Ενσωματωμένη δυνατότητα για αυτόματη παραγωγή XML documentation.

- Αυτόματο καθάρισμα της δυναμικά δεσμευμένης μνήμης.

- Δυνατότητα για μαρκάρισμα κλάσεων ή μεθόδων με κάποια συγκεκριμένα χαρακτηριστικά. Αυτό μπορεί να είναι χρήσιμο για την τεκμηρίωση της εφαρμογής αλλά και στην μεταγλώττιση της εφαρμογής. (για παράδειγμα μπορούμε να μαρκάρουμε κάποιες μεθόδους ώστε να μεταγλωττίζονται μόνο όταν η εφαρμογή είναι σε λειτουργία αποσφαλμάτωσης (debug mode)).

- Πλήρης πρόσβαση στη βασική βιβλιοθήκη του .NET Framework αλλά και εύκολη πρόσβαση στο Windows API (αν χρειαστεί).

- Προσπελαση της μνήμης με δείκτες ή απ' ευθείας (αν χρειαστεί). Γενικά η γλώσσα έχει σχεδιαστεί με τέτοιο τρόπο ώστε να μπορεί να λειτουργεί χωρίς

αυτά τα χαρακτηριστικά στις περισσότερες των περιπτώσεων.

- Υποστήριξη για ιδιότητες (properties) και γεγονότα (events) στο στυλ της Visual Basic.

- Αλλάζοντας τις επιλογές του μεταγλωττιστή, δίνεται η δυνατότητα να μεταγλωττίσουμε το προγραμμά μας σε ένα εκτελέσιμο αρχείο ή μια βιβλιοθήκη η οποία θα περιέχει .NET συστατικά τα οποία θα μπορούν να καλούνται από άλλα τμήματα κώδικα σαν ActiveX controls. (COM components)

- Η C# μπορεί να χρησιμοποιηθεί για συγγραφή α) δυναμικών ιστοσελίδων ASP.NET αλλά και β) XML Web Services

- Η C# υποστηρίζει έναν αυστηρό τύπο δεδομένων για boolean μεταβλητές, τον bool. Δηλώσεις οι οποίες έχουν κάποιες καταστάσεις όπως οι βρόγχοι while και if, απαιτούν μια έκφραση της οποίας ο προκύπτον τύπος θα είναι bool. Με απλά λόγια η έκφραση να μπορεί να είναι αληθής ή ψευδής. Ενώ η C++, υποστηρίζει boolean τύπους δεδομένων, αυτοί μπορούν εύκολα να μετατραπούν από/σε ακέραιους και εκφράσεις όπως οι if(a) απαιτούν απλά το a να μετατραπεί σε boolean τύπο, επιτρέποντας έτσι το a να είναι πχ. ακέραιος ή δείκτης. Η C# δεν επιτρέπει έναν ακέραιο να δείχνει αλήθεια ή ψεύδος οπότε αναγκάζει τους προγραμματιστές να χρησιμοποιούν εκφράσεις οι οποίες επιστρέφουν ακριβώς boolean τύπους δεδομένων. Ως αποτέλεσμα αυξάνεται η αποδοτικότητα και μειώνεται η πιθανότητα άσκοπων αλλά κοινών λαθών στον κώδικα. Ένα τέτοιο πρόβλημα στη C++ περιγράφεται με το εξής παράδειγμα: Όταν έχουμε μια έκφραση if(a=b) και χρησιμοποιούμε = αντί για ==.

- Σε αντίθεση με την δομή try...catch που χρησιμοποιούμε για χειρισμό λαθών, υπάρχει η δυνατότητα χρήσης της δομής try...finally η οποία εγγυάται εκτέλεση του κώδικα που βρίσκεται μέσα στο finally μπλοκ.

- Παρέχεται πλήρης δυνατότητα για reflection programming.

- Η C# (στην έκδοση 4.0) έχει 77 δεσμευμένες λέξεις.

Τα περισσότερα από τα παραπάνω χαρακτηριστικά τα έχουν και οι γλώσσες προγραμματισμού Visual Basic .NET και C++. Το γεγονός όμως το ότι η C# σχεδιάστηκε για χρήση των δυνατοτήτων του .NET Framework την κάνει πιο πλήρη, και προσφέρεται στους προγραμματιστές με πιο απλή σύνταξη σε σχέση με τις άλλες δύο.

Μειονεκτήματα

Πέρα από τη δύναμη της C#, υπάρχουν και κάποιοι περιορισμοί οι οποίοι την κάνουν ακατάλληλη γλώσσα προγραμματισμού για τη συγγραφή κάποιων εφαρμογών. Το κύριο μειονέκτημά της είναι ότι δεν έχει σχεδιαστεί για τη σύγγραφη προγραμμάτων τα οποία έχουν σαν πρώτη προτεραιότητα τις ακραίες επιδόσεις. Αν

λοιπόν ενδιαφέρει τον προγραμματιστή αν ένα βρογχος θα πάρει 1.050 κύκλους μηχανής αντί για 1.000, και αν κάθε δέκατο του δευτερολέπτου είναι σημαντικό για την ανάγκη που εξυπηρετεί μια εφαρμογή τότε η καλύτερη λύση μεταξύ των low-level γλωσσών παραμένει η C++.

Παρ'όλα αυτά το σύνολο των εφαρμογών που ανήκουν σε αυτή την κατηγορία είναι πολύ μικρό.

Υποστήριξη για Versioning

Η διαδικασία ανανέωσης ενός ήδη υπάρχοντος λογισμικού είναι επίπονη και επιρρεπής σε λάθη. Οι αλλαγές στον κώδικα μιας εφαρμογής μπορεί να αλλάξουν τον τρόπο λειτουργίας της. Η C# δίνει λύση σε αυτή τη διαδικασία καθώς περιλαμβάνει υποστήριξη για versioning.

Συγκεκριμένα, πολλές εφαρμογές που τρέχουν σε Windows παρουσιάζουν προβλήματα όταν για κάποιο λόγο μια assembly που χρησιμοποιούν αντικατασταθεί από μία νεότερη. Το πρόβλημα αυτό είναι ιδιαίτερα συχνό στο κόσμο των προγραμματιστών και “μπελάς” για τον κόσμο των εταιριών καθώς αυξάνουν τα κόστη ανάπτυξης. Στο .NET Framework κάθε DLL αρχείο ή COM αντικείμενο περιέχει πληροφορία για την έκδοσή του.

Ως αποτέλεσμα κάθε εφαρμογή που έχει χτιστεί πάνω στο .NET Framework φορτώνει τις εκδόσεις των DLLs ή COM αντικειμένων με τις οποίες έχει δοκιμαστεί ότι τρέχει χωρίς σφάλματα. Με τη βοήθεια αυτού του χαρακτηριστικού, επιτρέπεται σε σύνθετα πακέτα λογισμικού να αναπτύσσονται και να εξελίσσονται σε βάθος χρόνου κάνοντας τις νεότερες εκδόσεις του πιο σταθερές.

Web Programming

Εκείνο το χαρακτηριστικό της C# για το οποίο αξίζει να αφιερωθεί μια ενότητα είναι η πλήρης υποστήριξη που έχει για τα πιο νέα πρότυπα και πρωτόκολλα που υποστηρίζονται από τον παγκόσμιο ιστό. Η C# παρέχει υποστήριξη για τη μετατροπή οποιουδήποτε τμήματος κώδικα σε Web Service με αποτέλεσμα, συναρτήσεις και γενικότερα πολλές απο τις λειτουργίες του προγράμματος να μπορούν να προσπελαστούν από άλλες πλατφόρμες προγραμματισμού. Ο τεχνικός τρόπος με τον οποίο γίνεται αυτό είναι ο εξής: ένα XML Web Service ουσιαστικά είναι μια ASP.NET σελίδα η οποία επιστρέφει XML κώδικα στον πελάτη αντί για HTML όταν ο τελευταίος ζητήσει μια πληροφορία. Τέτοια προγράμματα παράγουν DLL αρχεία τα οποία περιέχουν κλάσεις οι οποίες προέρχονται από την WebService κλάση του .NET Framework. Για να γίνει κατανοητό το μέγεθος της σημαντικότητας αυτού του χαρακτηριστικού απλά αναφέρουμε τα XML Web Services στηρίζονται στο HTTP πρωτόκολλο. Τα XML Web Services μπορούν να χρησιμοποιηθούν πάνω από HTTP δίκτυα δεδομένου το ότι είναι ένα μέσο για μετάδοση πληροφοριών.

Η XML είναι μια γλώσσα αυτοπεριγραφής και μη ειδικευμένη η οποία είναι κατανοητή από όλες τις γλώσσες προγραμματισμού. Αυτός ο λόγος δίνει τη δυνατότητα στις εταιρίες, οι οποίες χρησιμοποιούν αυτή τη τεχνολογία, να

κατασκευάζουν μεγάλες εφαρμογές οι οποίες είναι ανεξάρτητες πλατφόρμας.

Σύνοψη

Η C# είναι μια μοντέρνα αντικειμενοστρεφής γλώσσα που δινει την δυνατότητα στους προγραμματιστές να φτιάχνουν γρήγορα και εύκολα λογισμικό για την πλατφόρμα .NET. Η παρεχόμενη πλατφόρμα επιτρέπει την κατασκευή XML Web Services τα οποία είναι διαθέσιμα μέσω Internet από οποιαδήποτε εφαρμογή που τρέχει σε οποιαδήποτε πλατφόρμα.

Η γλώσσα ενισχύει την παραγωγικότητα των υπευθύνων για την ανάπτυξη λογισμικού ενώ παράλληλα βοηθά τον προγραμματιστή στην μείωση των λαθών κατά τη φάση της υλοποίησης. Με αυτό το τρόπο αποτρέπει τις αυξανόμενες δαπάνες της ανάπτυξης λογισμικού. Τέλος, η C# βοηθά τη μετάβαση των C/C++ προγραμματιστών στην ανάπτυξη εφαρμογών για το παγκόσμιο ιστό ενώ παράλληλα διατηρεί τη δύναμη και την ευελιξία στην ανάπτυξη λογισμικού.

2.4 LINQ-Queries στη C#

Εισαγωγή στα Linq-Queries

Ένα ερώτημα είναι μια έκφραση που ανακτά δεδομένα από ένα αρχείο προέλευσης δεδομένων. Τα ερωτήματα είναι συνήθως εκφρασμένα σε εξειδικευμένη γλώσσα ερωτήματος. Διαφορετικές γλώσσες έχουν αναπτυχθεί διαχρονικά για διάφορους τύπους δεδομένων προέλευσης, για παράδειγμα η SQL για σχεσιακές βάσεις δεδομένων και η XQuery για XML. Ως εκ τούτου, οι προγραμματιστές έπρεπε να μάθουν μια νέα γλώσσα ερωτήματος για κάθε τύπο προέλευσης δεδομένων ή τη μορφή δεδομένων που πρέπει να υποστηρίξουν. Η LINQ απλοποιεί την κατάσταση αυτή, προσφέροντας ένα συνεπές μοντέλο για εργασία με δεδομένα σε διάφορα είδη πηγών δεδομένων και μορφές. Σε ένα ερώτημα LINQ, εργάζεστε πάντα με αντικείμενα. Μπορείτε να χρησιμοποιήσετε τα ίδια βασικά μοτίβα κωδικοποίησης στο ερώτημα και τα δεδομένα να μετασχηματίζονται σε έγγραφα XML, βάσεις δεδομένων SQL, σύνολα δεδομένων ADO.NET, .NET συλλογές, και οποιαδήποτε άλλη μορφή για την οποία παρέχεται LINQ provider.

Τρία μέρη για τη λειτουργία του ερωτήματος

Όλες οι λειτουργίες του ερωτήματος LINQ αποτελούνται από τρεις διακριτές ενέργειες:

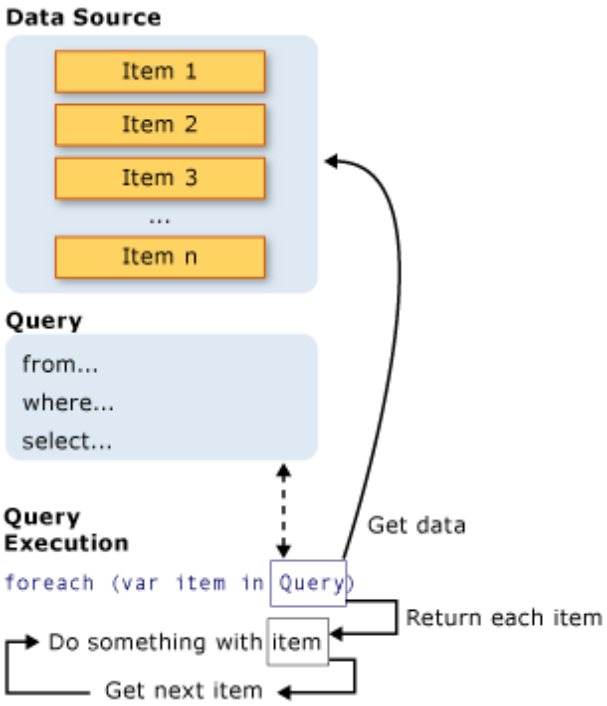
- Να αποκτήσετε το αρχείο προέλευσης δεδομένων.
- Να δημιουργήσετε το ερώτημα.
- Εκτέλεση του ερωτήματος.

Το παρακάτω παράδειγμα δείχνει πώς τα τρία μέρη μιας λειτουργίας ερωτήματος εκφράζονται σε πηγαίο κώδικα. Το παράδειγμα χρησιμοποιεί έναν πίνακα ακεραίων ως αρχείο προέλευσης δεδομένων για ευκολία. Ωστόσο, ισχύουν οι ίδιες έννοιες σε άλλες πηγές δεδομένων επίσης. Αυτό το παράδειγμα αναφέρεται σε όλο το υπόλοιπο αυτού του θέματος.

```
class IntroToLINQ
{
    static void Main()
    {
        // The Three Parts of a LINQ Query:
        //  1. Data source.
        int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

        // 2. Query creation.
        // numQuery is an IEnumerable<int>
        var numQuery =
            from num in numbers
```

```
            where (num % 2) == 0
            select num;

        // 3. Query execution.
        foreach (int num in numQuery)
        {
            Console.Write("{0,1} ", num);
        }
    }
}
```

Η παρακάτω εικόνα παρουσιάζει τη πλήρη λειτουργία ενός ερωτήματος. Στη LINQ διακρίνεται η εκτέλεση του ερωτήματος από το ίδιο το ερώτημα, με άλλα λόγια δεν γίνεται ανάκτηση δεδομένων απλά δημιουργώντας μια μεταβλητή ερωτήματος.



The Data Source

Στο προηγούμενο παράδειγμα, επειδή το αρχείο προέλευσης δεδομένων είναι ένας πίνακας, σιωπηρά υποστηρίζει το IEnumerable<T> interface. Το γεγονός αυτό σημαίνει ότι αυτό μπορεί να τεθεί ερώτημα με LINQ. Ένα ερώτημα εκτελείται σε μια πρόταση foreach, και η foreach απαιτεί IEnumerable ή IEnumerable <T>. Τύποι που υποστηρίζουν IEnumerable <T>ή ένα παραγόμενο interface όπως το generic IQueryable <T>ονομάζονται queryable τύποι. Ένας queryable τύπος δεν απαιτεί καμία τροποποίηση ή ειδική μεταχείριση για να χρησιμεύσει ως πηγή δεδομένων σε LINQ. Εάν τα δεδομένα προέλευσης δεν είναι ήδη στη μνήμη ως queryable τύπος, ο LINQ

provider πρέπει να τα εκπροσωπήσει ως τέτοια. Για παράδειγμα, LINQ σε XML φορτώνει ένα έγγραφο XML σε έναν queryable XElement τύπο:

```
// Create a data source from an XML document.
// using System.Xml.Linq;
XElement contacts = XElement.Load(@"c:\myContactList.xml");
```

Με LINQ σε SQL, πρώτα δημιουργείται ένα σχεσιακό-αντικείμενο χαρτογράφησης κατά το χρόνο σχεδίασης είτε με μη αυτόματο τρόπο ή χρησιμοποιώντας τον σχεδιαστή (O/R) στο Visual Studio. Μπορείτε να γράψετε τα ερωτήματα σας για τα αντικείμενα, και κατά το χρόνο εκτέλεσης η LINQ σε SQL χειρίζεται την επικοινωνία με τη βάση δεδομένων. Στο παρακάτω παράδειγμα, το Customers αντιπροσωπεύει ένα συγκεκριμένο πίνακα στη βάση δεδομένων, και ο τύπος του αποτέλεσματος του ερωτήματος, IQueryable <T>, προέρχεται από IEnumerable <T>.

```
Northwnd db = new Northwnd(@"c:\northwnd.mdf");

// Query for customers in London.
IQueryable<Customer> custQuery =
      from cust in db.Customers
      where cust.City == "London"
      select cust;
```

The Query

Το ερώτημα καθορίζει ποιες πληροφορίες θα ανακτηθούν από την πηγή/ πηγές δεδομένων. Προαιρετικά, ένα ερώτημα μπορεί να καθορίζει τον τρόπο που οι πληροφορίες να είναι ταξινομημένες, ομαδοποιημένες και σχηματισμένες προτού να επιστραφούν. Ένα ερώτημα αποθηκεύεται σε μια μεταβλητή ερωτήματος και αρχικοποιείται με μια έκφραση ερωτήματος. Για να γίνει ευκολότερη η γραφή ερωτημάτων,η C# έχει εισαγάγει νέα σύνταξη ερωτήματος.

Το ερώτημα στο προηγούμενο παράδειγμα επιστρέφει όλους τους ζυγούς αριθμούς από το πίνακα ακεραίων. Η έκφραση ερωτήματος περιέχει τρεις προτάσεις: from, where και select. Η πρόταση from καθορίζει το αρχείο προέλευσης δεδομένων, η where καθορίζει πού εφαρμόζεται το φίλτρο, και το select καθορίζει τον τύπο των στοιχείων που επιστρέφονται. Αυτές και άλλες προτάσεις ερωτημάτων, που έχουν χρησιμοποιηθεί στη δική μας εφαρμογή συζητούνται λεπτομερώς στη συνέχεια.

Query Execution

Όπως αναφέρθηκε προηγουμένως,η ίδια η μεταβλητή ερωτήματος αποθηκεύει μόνο τις εντολές ερωτήματος. Την πραγματική εκτέλεση του ερωτήματος αναβάλλεται μέχρι να επαναλάβεις τη μεταβλητή ερωτήματος σε μια δήλωση foreach. Η έννοια αυτή αναφέρεται ως αναστολή εκτέλεσης και αποδεικνύεται στο ακόλουθο παράδειγμα:

```
//  Query execution.
foreach (int num in numQuery)
{
    Console.Write("{0,1} ", num);
}
```

Η δήλωση foreach είναι επίσης όπου ανακτώνται τα αποτελέσματα του ερωτήματος. Για παράδειγμα, στο προηγούμενο ερώτημα, η μεταβλητή επανάληψης num κρατάει κάθε τιμή (μία κάθε φορά) για την ακολουθία που επιστρέφεται. Επειδή η ίδια η μεταβλητή ερωτήματος ποτέ δεν κρατάει τα αποτελέσματα του ερωτήματος, μπορείτε να την εκτελείτε όσο συχνά θέλετε. Για παράδειγμα, μπορεί να έχετε μια βάση δεδομένων που ενημερώνεται συνεχώς από μια ξεχωριστή εφαρμογή. Στην εφαρμογή σας, θα μπορούσατε να δημιουργήσετε ένα ερώτημα που ανακτά τα πιο πρόσφατα δεδομένα, και θα μπορούσατε να το εκτελείτε επανειλημμένα σε κάποιο διάστημα για να ανακτάτε διαφορετικά αποτελέσματα κάθε φορά.

Ερωτήματα που εκτελούν λειτουργίες συνάθροισης σε μια σειρά από στοιχεία προέλευσης πρώτα πρέπει να επαναλαμβάνονται σε αυτά τα στοιχεία. Παραδείγματα τέτοιων ερωτημάτων είναι Count, Max,Average, και First. Αυτά εκτελούνται χωρίς δήλωση foreach επειδή το ίδιο το ερώτημα, πρέπει να χρησιμοποιήσει foreach προκειμένου να επιστρέψει ένα αποτέλεσμα. Σημειώστε επίσης ότι αυτοί οι τύποι των ερωτημάτων επιστρέφουν μία μόνο τιμή, και όχι μια συλλογή IEnumerable. Το παρακάτω ερώτημα επιστρέφει μια καταμέτρηση των ζυγών αριθμών του πίνακα προέλευσης:

```
var evenNumQuery =
      from num in numbers
      where (num % 2) == 0
      select num;

int evenNumCount = evenNumQuery.Count();
```

Για να επιβάλλετε την άμεση εκτέλεση οποιουδήποτε ερωτήματος και προσωρινή αποθήκευση των αποτελεσμάτων του, μπορείτε να καλέσετε τις ToList <TSource>ή ToArray <TSource>μεθόδους.

```
List<int> numQuery2 =
      (from num in numbers
       where (num % 2) == 0
      select num).ToList();

// or like this:
// numQuery3 is still an int[]

var numQuery3 =
      (from num in numbers
      where (num % 2) == 0
      select num).ToArray();
```

Προτάσεις που χρησιμοποιήθηκαν στην εφαρμογή

◆ *From* clause:

Ένα ερώτημα πρέπει να αρχίζει με μια πρόταση from. Επιπλέον,ένα ερώτημα μπορεί να περιέχει υπο-ερωτήματα, που επίσης να αρχίζουν με μια πρόταση from .
Η πρόταση from καθορίζει τα εξής:

- Το αρχείο προέλευσης δεδομένων στο οποίο θα εκτελεστεί το ερώτημα ή το υπο-ερώτημα.
- Μια μεταβλητή τοπικής εμβέλειας που αντιπροσωπεύει κάθε στοιχείο στην ακολουθία προέλευσης.

Τόσο η μεταβλητή όσο και το αρχείο προέλευσης δεδομένων είναι έντονα δακτυλογραφημένες. Το αρχείο προέλευσης δεδομένων που αναφέρεται σε μια πρόταση from, πρέπει να είναι τύπου IEnumerable, IEnumerable <T>ή παραγόμενος τύπος όπως IQueryable <T>.
Στο παρακάτω παράδειγμα,το numbers είναι το αρχείο προέλευσης δεδομένων και num είναι η τοπική μεταβλητή. Προσέξτε ότι και οι δύο μεταβλητές είναι έντονα δακτυλογραφημένες παρότι η λέξη-κλειδί var χρησιμοποιείται.

```
class LowNums
{
    static void Main()
    {
        // A simple data source.
        int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

        // Create the query.
        // lowNums is an IEnumerable<int>
        var lowNums = from num in numbers
            where num < 5
            select num;

        // Execute the query.
        foreach (int i in lowNums)
        {
            Console.Write(i + " ");
        }
    }
}
// Output: 4 1 3 2 0
```

The range variable:

Ο μεταγλωττιστής συνάγει το είδος της μεταβλητής, όταν το αρχείο προέλευσης δεδομένων υλοποιεί IEnumerable <T>. Για παράδειγμα, αν η πηγή είναι τύπου IEnumerable <Customer>, τότε η μεταβλητή συνάγεται να είναι Customer . Η μόνη φορά που πρέπει να καθοριστεί ξεχωριστά ο τύπος είναι όταν η πηγή είναι non-generic IEnumerable τύπου όπως ArrayList.

Στο προηγούμενο παράδειγμα η num συνάγεται να είναι του τύπου int. Επειδή η μεταβλητή είναι έντονα δακτυλογραφημένη, μπορείτε να καλέσετε μεθόδους σε αυτή ή να τη χρησιμοποιήσετε σε άλλες λειτουργίες. Για παράδειγμα, αντί να γράφετε select num, θα μπορούσατε να γράψετε select num. ToString () να προκαλέσετε το ερωτήμα να επιστρέψει μια ακολουθία συμβολοσειρών αντί για ακέραιους αριθμούς. Ή θα μπορούσατε να γράψετε, select n + 10 για να προκαλέσετε το ερώτημα να επιστρέψει την ακολουθία 14, 11, 13, 12, 10.

Η τοπική μεταβλητή είναι σαν μια μεταβλητή επανάληψης σε μια δήλωση foreach εκτός από μία πολύ σημαντική διαφορά: μια τοπική μεταβλητή στην πραγματικότητα ποτέ δεν αποθηκεύει δεδομένα από το αρχείο προέλευσης. Είναι απλά μια συντακτική ευκολία που επιτρέπει στο ερώτημα να περιγράψει τι θα συμβεί όταν εκτελεστεί το ερώτημα.

- ***Where* clause:**

Η πρόταση Where χρησιμοποιείται σε μια παράσταση ερωτήματος για να καθορίσει ποια στοιχεία του αρχείου προέλευσης δεδομένων θα επιστραφούν από το ερωτήμα. Εφαρμόζει μια δυαδική κατάσταση (predicate) για κάθε στοιχείο προέλευσης (που αναφέρεται από την τοπική μεταβλητή) και επιστρέφει εκείνες για τις οποίες η συγκεκριμένη συνθήκη είναι αληθής. Μια μόνο παράσταση ερωτήματος μπορεί να περιέχει πολλαπλές προτάσεις where και μία πρόταση ενδέχεται να περιέχει πολλαπλές predicate υποεκφράσεις.

Στο παρακάτω παράδειγμα,η πρόταση where φιλτράρει όλους τους αριθμούς εκτός από εκείνους που είναι λιγότερο από πέντε. Εάν καταργήσετε την πρόταση where, όλοι οι αριθμοί από το αρχείο προέλευσης δεδομένων θα επιστραφούν. Η έκφραση num < 5 είναι το predicate που εφαρμόζεται σε κάθε στοιχείο.

```
class WhereSample
{
    static void Main()
    {
        // Simple data source. Arrays support IEnumerable<T>.
        int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

        // Simple query with one predicate in where clause.
        var queryLowNums =
            from num in numbers
            where num < 5
            select num;
```

```
        // Execute the query.
        foreach (var s in queryLowNums)
        {
            Console.Write(s.ToString() + " ");
        }
    }
}
//Output: 4 1 3 2 0
```

Μέσα σε μια μόνο where πρόταση, μπορείτε να καθορίσετε όσα predicates σας χρειάζονται, χρησιμοποιώντας τους && και || τελεστές. Στο παρακάτω παράδειγμα, το ερώτημα καθορίζει δύο predicates προκειμένου να επιλέξετε μόνο τους ζυγούς αριθμούς που είναι μικρότεροι του πέντε.

```
// Create the query with two predicates in where clause.
var queryLowNums2 =
    from num in numbers
    where num < 5 && num % 2 == 0
    select num;
```

- ***Select* clause:**

Σε μια έκφραση ερωτήματος, η πρόταση select καθορίζει τον τύπο των τιμών που θα παραχθούν όταν εκτελεστεί το ερώτημα. Το αποτέλεσμα βασίζεται στην αξιολόγηση όλων των προηγούμενων προτάσεων και σε εκφράσεις στην ίδια την select πρόταση. Ένα ερώτημα πρέπει να τερματιστεί είτε με μια select πρόταση ή μια πρόταση group.

Το παρακάτω παράδειγμα δείχνει μια απλή πρόταση select σε μια παράσταση ερωτήματος.

```
class SelectSample1
{
    static void Main()
    {
        //Create the data source
        List<int> Scores = new List<int>() { 97, 92, 81, 60 };

        // Create the query.
        IEnumerable<int> queryHighScores =
            from score in Scores
            where score > 80
            select score;

        // Execute the query.
        foreach (int i in queryHighScores)
        {
            Console.Write(i + " ");
        }
    }
}
//Output: 97 92 81
```

Ο τύπος της ακολουθίας που παράγεται από την select πρόταση καθορίζει τον τύπο της μεταβλητής ερωτήματος queryHighScores. Στην απλούστερη περίπτωση,η select πρόταση καθορίζει ακριβώς την τοπική μεταβλητή. Αυτό προκαλεί την ακολουθία που επιστρέφεται να περιέχει στοιχεία ίδιου τύπου σαν αυτά του αρχείου προέλευσης δεδομένων. Ωστόσο, η πρόταση select παρέχει επίσης έναν ισχυρό μηχανισμό για μετατροπή (ή προβολής) των δεδομένων προέλευσης σε νέους τύπους.

◆ *Let* clause:

Σε μια έκφραση ερωτήματος, μερικές φορές είναι χρήσιμο να αποθηκεύσετε το αποτέλεσμα μιας υπο-έκφρασης προκειμένου να το χρησιμοποιήσετε σε επόμενες προτάσεις. Μπορείτε να το κάνετε με την λέξη let, η οποία δημιουργεί μια νέα τοπική μεταβλητή και την αρχικοποιεί με το αποτέλεσμα της έκφρασης που παρέχετε. Αφού αρχικοποιηθεί με μια τιμή, η τοπική μεταβλητή δεν μπορεί να χρησιμοποιηθεί για να αποθηκεύσει άλλη τιμή. Ωστόσο, αν η μεταβλητή κρατάει ένα queryable τύπο, μπορεί να γίνει ερώτημα.

Στο παρακάτω παράδειγμα η let χρησιμοποιείται με δύο τρόπους:

1. Για να δημιουργήσετε ένα enumerable τύπο που μπορεί ο ίδιος να τεθεί ως ερώτημα.

2. Για να ενεργοποιήσετε το ερώτημα να καλέσει την `ToLower` () μόνο μία φορά στην τοπική μεταβλητή `word` . Χωρίς τη χρήση της let, θα πρέπει να καλέσετε την `ToLower`() σε κάθε predicate της πρότασης where.

```
     class LetSample1
{
    static void Main()
    {
        string[] strings =
        {
            "A penny saved is a penny earned.",
            "The early bird catches the worm.",
            "The pen is mightier than the sword."
        };

        // Split the sentence into an array of words
        // and select those whose first letter is a vowel.
        var earlyBirdQuery =
            from sentence in strings
            let words = sentence.Split(' ')
            from word in words
            let w = word.ToLower()
```

```
            where w[0] == 'a' || w[0] == 'e'
                || w[0] == 'i' || w[0] == 'o'
                || w[0] == 'u'
            select word;

        // Execute the query.
        foreach (var v in earlyBirdQuery)
        {
            Console.WriteLine("\"{0}\" starts with a vowel", v);
        }

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
/* Output:
    "A" starts with a vowel
    "is" starts with a vowel
    "a" starts with a vowel
    "earned." starts with a vowel
    "early" starts with a vowel
    "is" starts with a vowel
*/
```

3

Τύποι αρχείων που χρησιμοποιήθηκαν στην εφαρμογή

3.1 Xml (eXtensive Markup Language) αρχεία

Γενικά

Η XML είναι μια markup γλώσσα για έγγραφα τα οποία περιέχουν δομημένη πληροφορία, και αποτελεί σημαντικό στοιχείο του semantic web. Η δομημένη πληροφορία περιέχει περιεχόμενα (λέξεις, εικόνες κτλ) και κάποια στοιχεία για το τί αυτή αντιπροσωπεύει, και τι ρόλο παίζει. Σχεδόν όλα τα έγγραφα έχουν κάποια δομή. Μια markup γλώσσα είναι ένας μηχανισμός ώστε να υποδηλώνεται η δομή ενός εγγράφου. Οι προδιαγραφές της XML ορίζουν ένα τυποποιημένο τρόπο με τον οποίο μπορεί κάποιος να προσθέτει markup στα έγγραφα. Ένα τυπικό παράδειγμα στο οποίο μπορούμε να δούμε μια πληροφορία και το είδος το οποίο αντιπροσωπεύει είναι το εξής:

<author>Berners-Lee</author>.

Το πλεονέκτημα της XML είναι ότι το λογισμικό μπορεί να διαβάσει συγκεκριμένες ετικέτες και να εφαρμόσει διαδικασίες όπως είναι η εξόρυξη βιβλιογραφικής πληροφορίας αλλά και άλλες χρήσιμες διαδικασίες. Οι σχεδιαστικοί στόχοι της XML δίνουν έμφαση στην απλότητα, τη γενικότητα και την χρήση της στο Internet. Η γλώσσα αυτή υποστηρίζεται από όλες τις σύγχρονες γλώσσες προγραμματισμού του κόσμου και παρ' όλο το ότι στοχεύει στη δόμηση εγγράφων, υπάρχουν πολλές περιπτώσεις στις οποίες χρησιμοποιείται για την παρουσίαση πολύπλοκων δομών δεδομένων όπως για παράδειγμα είναι τα XML Web Services.

Τέλος,πρέπει να αναφερθεί ότι η XML είναι η βάση για ένα σύνολο άλλων γλωσσών όπως οι: RSS, Atom, SOAP, XHTML και άλλες.

Ορισμός της δομής ενός XML εγγράφου

Όπως αναφέρθηκε προηγουμένως, η XML είναι ένα έγγραφο με καθορισμένη δομή. Αυτή η δομή όμως πως ορίζεται; Παρακάτω θα δωθούν πληροφορίες για το XML Schema το οποίο είναι ένας τρόπος για να περιγράψουμε τη δομή ενός αρχείου XML.

Το XML Schema λοιπόν εκφράζει με ένα κοινό λεξιλόγιο το περιεχόμενο του XML εγγράφου. Πιο συγκεκριμένα, παρέχει τα μέσα ώστε να δηλωθεί η δομή το περιεχόμενο και τα tags ενός XML αρχείου. Αυτή η διαδικασία επικυρώνει τη σύνταξη και τη φόρμα που επιτρέπεται σε κάθε τέτοιο έγγραφο και διευκολύνει το διαμοιρασμό πληροφοριών μέσω διαδικτύου. Το XML Schema επιτρέπει το διάβασμα, την επικύρωση και την επεξεργασία XML εγγράφων από εφαρμογές λογισμικού. Αυτό παρέχει την βάση για τη σύλληψη, αναπαράσταση, ανταλλαγή και αποθήκευση πληροφοριών στις οποίες μπορούν εύκολα να έχουν πρόσβαση ευφυείς πράκτορες.

XML έννοιες

(Unicode) Character:

Από τον ορισμό της γλώσσας, είναι γνωστό ότι το έγγραφο XML είναι ένα αλφαριθμητικό το οποίο αποτελείται απο χαρακτήρες. Σχεδόν κάθε Unicode χαρακτήρας μπορεί να εμφανιστεί σε ένα XML έγγραφο.

Processor and Application:

Ο επεξεργαστής αναλύει το markup κομμάτι του εγγράφου και παραχωρεί σε μια εφαρμογή τη δομημένη πληροφορία που λαμβάνει από αυτό. Οι προδιαγραφές τις γλώσσες ορίζουν τι πρέπει να κάνει ένας επεξεργαστής και τι όχι. Από τη σκοπιά του προγράμματος, δεν υπάρχουν προδιαγραφές για το πως θα διαχειριστούν τα δεδομένα. Ο επεξεργαστής συχνά αναφέρεται και ως XML parser.

Markup and Content:

Οι χαρακτήρες οι οποία αποτελούν ένα XML έγγραφο είναι διαμοιρασμένοι σε markup και περιεχόμενο. Αυτά τα δύο πρέπει να διαχωρίζονται από την εφαρμογή με κάποιους απλούς συντακτικούς κανόνες.
Παραδείγματος χάριν όλα τα αλφαριθμητικά τα οποία ανήκουν στη κατηγορία markup ξεκινάν ή τελειώνουν με τα σύμβολα “<” και “>” αντίστοιχα. Επίσης είναι δυνατό να ξεκινούν με το χαρακτήρα “&” και να τελειώνουν με το χαρακτήρα “;”. Τα αλφαριθμητικά τα οποία δεν αποτελούν markup είναι το περιεχόμενο.

Ετικέτες (Tags):

Ετικέτα ονομάζεται το markup αλφαριθμητικό το οποίο ξεκινά με τον χαρακτήρα “<” και τελειώνει με το χαρακτήρα “>”. Τα tags αποτελούνται απο τρεις κατηγορίες.

Αρχικά υπάρχουν τα start-tags (ετικέτες αρχής), πχ.<section>, στη συνέχεια έχουμε τα end-tags (ετικέτες τέλους), πχ.</section> και τέλος έχουμε τα empty-element tags , πχ. <line-break/>.

Στοιχείο (Element):

Το στοιχείο είναι κάθε λογικό συστατικό του εγγράφου το οποίο είτε ξεκινά με ένα start-tag και τελειώνει με ένα end-tag, είτε αποτελείται από ένα empty-element tag. Οι χαρακτήρες ανάμεσα από τα start-tags και end-tags,αν υπάρχουν, είναι το περιεχόμενο του στοιχείου, και πιθανώς να περιέχουν markup στοιχεία αλλά και άλλα στοιχεία τα οποία καλούνται elements-παιδιά.

Για παράδειγμα το <Greeting>Hello, world.</Greeting>.

Ιδιότητες (Attribute):

Ιδιότητα ονομάζεται μια markup δομή η οποία αποτελείται από ένα ζευγάρι ονόματος-τιμής το οποίο βρίσκεται μέσα σε ένα start-tag ή empty element tag. Για παράδειγμα, παρακάτω το στοιχείο img έχει δύο ιδιότητες, src και alt:

.

Άλλο έναπαράδειγμα θα ήταν το

<step number=”3”>Connect A to B.</step>

όπου το όνομα της ιδιότητας είναι το “number” και ο αριθμός είναι το “3”.

XML δήλωση (XML Declaration):

Στα XML έγγραφα πρέπει να δηλώνονται στην αρχή κάποιες πληροφορίες γύρω από αυτά. Για παράδειγμα ποια έκδοση από το πρότυπο XML χρησιμοποιούν και σε τι κωδικοποίηση είναι γραμμένα αυτά τα έγγραφα.

Σύνοψη

Η XML είναι μια markup γλώσσα η οποία έχει δημιουργηθεί για την ανταλλαγή πληροφοριών στο διαδίκτυο έυκολα και αποδοτικά. Το μεγάλο πλεονέκτημά της είναι η δυνατότητα προσπέλασης των XML εγγράφων από οποιαδήποτε πλατφόρμα λογισμικού η υλικού. Παράλληλα είναι μια γλώσσα εύκολη στην κατανόηση και μπορεί να βοηθήσει στην προσπέλαση δεδομένων από άτομα με ειδικές ανάγκες.

Τέλος, η XML αποτελεί τη βάση δημιουργίας μιας πλειάδας γλωσσών που χρησιμοποιούνται στο Internet και χρησιμοποιείται κατα κόρον στα Web Services.

3.2 Csv (Comma- Separated Values) αρχεία

Ένα CSV αρχείο αποθηκεύει πινακοειδή δεδομένα (αριθμούς και κείμενο) σε απλό κείμενο. Κάθε γραμμή του αρχείου είναι μια εγγραφή δεδομένων. Κάθε εγγραφή αποτελείται από ένα ή περισσότερα πεδία, διαχωριζόμενα με κόμμα. Η χρήση κόμματος ως διαχωριστή πεδίων είναι η πηγή του ονόματος αυτής της μορφής αρχείου.

Στη δημοφιλή χρήση, εντούτοις, ο όρος CSV μπορεί να σημαίνει μερικές στενα συνδεδεμένες οριοθέτες- χωρισμένες μορφές,οι οποίες χρησιμοποιούν ποικίλους διαφορετικούς οριοθέτες πεδίων. Αυτές περιαμβάνουν tab- seperated τιμές και space-seperated τιμές , που και οι δυο από τις οποίες είναι δημοφιλείς. Σε τέτοια αρχεία συχνά δίνεται μια .csv επέκταση, παρά τη χρήση ενός διαφορετικού διαχωριστή πεδίων από το κόμμα. Αυτή η χαλαρή ορολογία δημιουργεί προβλήματα για την ανταλλαγή δεδομένων.

Ανταλλαγή δεδομένων

CSV είναι μια κοινή μορφή ανταλλαγής δεδομένων που υποστηρίζεται ευρέως από τον καταναλωτή, την επιχείρηση, και τις επιστημονικές εφαρμογές. Μεταξύ των πιό κοινών χρήσεών του είναι να μετακινεί τα πινακοειδή δεδομένα μεταξύ προγραμμάτων που λειτουργούν εγγενώς σε ασυμβίβαστες (συχνά ιδιόκτητες ή/και ατεκμηρίωτες) μορφές. Αυτό λειτουργεί παρά την έλλειψη συμμόρφωσης με το RFC 4180 (ή οποιοδήποτε άλλο πρότυπο), γιατί έτσι πολλά προγράμματα υποστηρίζουν παραλλαγές της μορφής CSV για εισαγωγή δεδομένων.

Για παράδειγμα, ένας χρήστης μπορεί να χρειαστεί να μεταφέρει πληροφορίες από ένα πρόγραμμα βάσης δεδομένων που αποθηκεύει δεδομένα σε ένα ιδιόκτητο μορφότυπο, σε ένα φύλλο που χρησιμοποιεί μια εντελώς διαφορετική μορφή. Το πρόγραμμα βάσης δεδομένων πιο πιθανό να μπορεί να εξάγει τα δεδομένα του ως "CSV". Το εξαγόμενο αρχείο CSV μπορεί στη συνέχεια να εισαχθεί από το πρόγραμμα υπολογιστικών φύλλων.

Προδιαγραφές

Το RFC 4180 προτείνει μια προδιαγραφή για τη μορφή CSV, και αυτός είναι ο ορισμός που χρησιμοποιείται συνήθως. Ωστόσο, σε λαϊκή χρήση "CSV" δεν είναι μια ενιαία, καλά προσδιορισμένη μορφή.

Ως αποτέλεσμα, στην πράξη, ο όρος "CSV" μπορεί να αναφέρεται σε οποιοδήποτε αρχείο που:

- είναι απλό κείμενο χρησιμοποιώντας ένα σύνολο χαρακτήρων όπως ASCII, διάφορα σύνολα χαρακτήρων Unicode (π.χ. UTF-8), EBCDIC, ή Shift JIS.
- αποτελείται από εγγραφές (συνήθως μια εγγραφή ανά γραμμή)
- με τις εγγραφές να χωρίζονται σε πεδία που διαχωρίζονται με διαχωριστικά (συνήθως ένα ενιαίο χαρακτήρα, όπως κόμμα, ερωτηματικό, ή tab -- μερικές φορές το διαχωριστικό μπορεί να περιλαμβάνει προαιρετικά διαστήματα)

- όπου κάθε εγγραφή έχει την ίδια ακολουθία πεδίων.

Μέσα σε αυτούς τους γενικούς περιορισμούς, πολλές παραλλαγές είναι σε χρήση. Ως εκ τούτου, χωρίς πρόσθετες πληροφορίες (όπως το αν το RFC 4180 τιμάται), ένα αρχείο που ισχυρίστηκε απλώς να είναι σε "CSV" μορφή δεν είναι πλήρως καθορισμένη. Ως αποτέλεσμα, πολλές εφαρμογές που υποστηρίζουν αρχεία CSV επιτρέπουν στους χρήστες να δούν τις πρώτες γραμμές του αρχείου και, στη συνέχεια, να καθορίσουν το διαχωριστικό χαρακτήρα. Αν παραλλαγές ενός συγκεκριμένου αρχείου CSV δεν εμπίπτουν σε ό,τι υποστηρίζει ένα συγκεκριμένο πρόγραμμα υποδοχής, είναι συχνά εφικτό να εξετάσεις και να επεξεργαστείς το αρχείο με το χέρι (δηλαδή, με έναν επεξεργαστή κειμένου) ή να γράψεις ένα σενάριο ή πρόγραμμα που να παράγουν μια σύμφωνη μορφή.

Γενική λειτουργικότητα

Οι CSV μορφές χρησιμοποιούνται καλύτερα για να αντιπροσωπεύουν αλληλουχίες ή σύνολα εγγραφών στις οποίες η κάθε εγγραφή έχει την ίδια λίστα πεδίων. Αυτό αντιστοιχεί σε μια ενιαία σχέση σε μια σχεσιακή βάση δεδομένων, ή σε δεδομένα (όχι υπολογισμούς) ενός υπολογιστικού φύλλου. Η μορφή χρονολογείται από τις πρώτες ημέρες των επιχειρήσεων πληροφορικής και χρησιμοποιείται ευρέως για τη μεταφορά δεδομένων μεταξύ υπολογιστών με διαφορετικά εσωτερικής λέξης μεγέθη, τις ανάγκες μορφοποίησης δεδομένων, και ούτω καθεξής. Για το λόγο αυτό, τα αρχεία CSV είναι κοινά από όλες τις πλατφόρμες υπολογιστών.

CSV είναι ένα οριοθετημένο αρχείο κειμένου που χρησιμοποιεί ένα κόμμα για να διαχωρίσει τις τιμές του (πολλές υλοποιήσεις εργαλείων εισαγωγής/εξαγωγής CSV επιτρέπουν να χρησιμοποιούνται και άλλοι διαχωριστές). Απλές εφαρμογές CSV μπορεί να απαγορεύουν τιμές πεδίων που περιέχουν ένα κόμμα ή άλλους ειδικούς χαρακτήρες, όπως νέες γραμμές. Πιο εξελιγμένες εφαρμογές CSV επιτρέπουν τέτοιους χαρακτήρες, συχνά απαιτώντας " (διπλά εισαγωγικά) γύρω από τις τιμές που περιέχουν δεσμευμένους χαρακτήρες. Τα ενσωματωμένα διπλά εισαγωγικά μπορεί στη συνέχεια να αντιπροσωπεύονται από ένα ζεύγος διαδοχικών διπλών εισαγωγικών, ή με το πρόθεμα ενός χαρακτήρα διαφυγής όπως το backslash.

Οι CSV μορφές δεν περιορίζονται σε ένα συγκεκριμένο σύνολο χαρακτήρων. Θα λειτουργήσουν εξίσου καλά με τα σετ χαρακτήρων Unicode (όπως UTF-8 ή UTF-16) και με ASCII (αν και ιδίως προγράμματα που υποστηρίζουν CSV ενδέχεται να έχουν τους δικούς τους περιορισμούς) . Τα CSV αρχεία κανονικά θα επιβιώσουν ακόμη και σε μια μετάφραση από το ένα σύνολο χαρακτήρων σε άλλο (σε αντίθεση με σχεδόν όλες τις αποκλειστικές μορφές δεδομένων). Το CSV δεν παρέχει, ωστόσο, κανένα τρόπο που να δείχνει ποιό σύνολο χαρακτήρων είναι σε χρήση, έτσι ώστε να πρέπει να κοινοποιούνται χωριστά, ή να καθορίζεται από τον παραλήπτη (αν είναι δυνατόν).
Οι βάσεις δεδομένων που περιλαμβάνουν πολλαπλές σχέσεις, δεν μπορούν να εξαχθούν ως ένα ενιαίο αρχείο CSV .

Ομοίως, το CSV δεν μπορεί να εκπροσωπεί ιεραρχική ή αντικειμενοστραφή βάση δεδομένων ή άλλα δεδομένα. Και αυτό συμβαίνει γιατί κάθε CSV εγγραφή αναμένεται

να έχει την ίδια δομή. Ως εκ τούτου, CSV είναι σπάνια κατάλληλη για έγγραφα όπως αυτά που δημιουργήθηκαν με την HTML, XML, ή άλλες τεχνολογίες σήμανσης ή επεξεργασίας κειμένου.

Οι στατιστικές βάσεις δεδομένων σε διάφορους τομείς έχουν συχνά μια γενική δομή που μοιάζει με σχέση, αλλά με ορισμένες επαναλαμβανόμενες ομάδες πεδίων. Για παράδειγμα, οι βάσεις δεδομένων για την υγεία, όπως η Δημογραφική και η Έρευνα Υγείας επαναλάμβάνει συνήθως μερικές ερωτήσεις για κάθε παιδί ενός δεδομένου γονέα (ίσως μέχρι ένα καθορισμένο μέγιστο αριθμό παιδιών). Τα συστήματα στατιστικής ανάλυσης συχνά περιλαμβάνουν βοηθητικά προγράμματα που μπορούν να "περιστρέφουν" αυτά τα δεδομένα. Για παράδειγμα, μια εγγραφή “γονέας” που περιλαμβάνει πληροφορίες για πέντε “παιδιά” μπορεί να χωριστεί σε πέντε ξεχωριστές εγγραφές, που η καθεμιά περιέχει

(α) τις πληροφορίες για ένα παιδί, και

(β)ένα αντίγραφο όλων των πληροφοριών που δεν αφορούν τα παιδιά.

Το CSV μπορεί να αντιπροσωπεύει είτε την "κάθετη" ή την "οριζόντια" μορφή των δεδομένων αυτών.

Σε μια σχεσιακή βάση δεδομένων, παρόμοια θέματα αντιμετωπίζονται εύκολα, δημιουργώντας μια ξεχωριστή σχέση, για κάθε τέτοια ομάδα, συνδέοντας τις εγγραφές "παιδία" με τις σχετικές εγγραφές “γονείς” χρησιμοποιώντας ένα ξένο κλειδί (όπως ο αριθμός ταυτότητας ή το όνομα του γονέα). Σε γλώσσες σήμανσης όπως η XML, τέτοιες ομάδες περικλείονται μέσα σε ένα στοιχείο του γονέα και επαναλαμβάνονται όταν χρειάζεται (για παράδειγμα, πολλοί <child> κόμβοι σε ένα ενιαίο <parent >κόμβο). Με τα CSV δεν υπάρχει ευρέως αποδεκτή λύση ενός αρχείου.

Τυποποίηση

Η ονομασία "CSV" υποδεικνύει τη χρήση κόμματος για να διαχωρίζονται πεδία δεδομένων. Παρ 'όλα αυτά, ο όρος «CSV» χρησιμοποιείται ευρέως για να παραπέμψει σε μια μεγάλη οικογένεια μορφών, που διαφέρουν με πολλούς τρόπους. Ορισμένες εφαρμογές επιτρέπουν ή απαιτούν μονά ή διπλά εισαγωγικά γύρω από ορισμένα ή όλα τα πεδία, και μερικές διατηρούν την πρώτη εγγραφή ως επικεφαλίδα περιέχοντας μια λίστα με τα ονόματα των πεδίων. Το σετ χαρακτήρων που χρησιμοποιείται είναι απροσδιόριστο: ορισμένες εφαρμογές απαιτούν ένα Unicode byte order mark(BOM) για την επιβολή Unicode ερμηνείας. Τα αρχεία που χρησιμοποιούν το χαρακτήρα tab αντί για κόμμα μπορεί να αναφέρονται με μεγαλύτερη ακρίβεια ως “TSV" για τιμές διαχωρισμένες με tab.

Άλλες διαφορές εφαρμογής περιλαμβάνουν τον χειρισμό των πιο σύνηθισμένων διαχωριστών πεδίων (όπως το διάστημα ή άνω τελεία) και ο χαρακτήρας αλλαγής γραμμής μέσα σε πεδία κειμένου. Μια ακόμα λεπτότητα είναι η ερμηνεία της κενής γραμμής: μπορεί εξίσου να είναι το αποτέλεσμα της συγγραφής μιας εγγραφής με μηδέν πεδία, ή μιας εγγραφής ένος πεδίου μηδενικού μήκους, έτσι η αποκωδικοποίηση είναι διφορούμενη.

Η εξάρτηση από το πρότυπο που τεκμηριώνεται από το RFC 4180 μπορεί να απλοποιήσει την CSV ανταλλαγή. Ωστόσο, το πρότυπο αυτό καθορίζει μόνο το χειρισμό των πεδίων κειμένου. Η ερμηνεία του κειμένου του κάθε πεδίου είναι ακόμα συγκεκριμένης εφαρμογής. Το RFC 4180 επισημοποιήθηκε ως CSV. Καθορίζει τον τύπο

MIME "text/ CSV", και τα αρχεία CSV που ακολουθούν τους κανόνες του θα πρέπει να είναι πολύ ευρέως φορητά. Μεταξύ των υποχρεώσεών τους είναι:

- Οι γραμμές τύπου MS-DOS που τελειώνουν με (CR / LF) χαρακτήρες (προαιρετικοί για την τελευταία γραμμή)

- Μια προαιρετική εγγραφή κεφαλίδας (δεν υπάρχει κανένας σίγουρος τρόπος για να εξακριβώσουμε αν υπάρχει, έτσι απαιτείται προσοχή κατά την εισαγωγή).

- Κάθε εγγραφή "θα πρέπει" να περιέχει τον ίδιο αριθμό διαχωρισμένων με κόμμα πεδίων.

- Κάθε πεδίο μπορεί να αναφέρεται (με διπλά εισαγωγικά).

- Τα πεδία που περιέχουν line-break, σε διπλά εισαγωγικά, και τα κόμματα θα πρέπει να έχουν διπλά εισαγωγικά. (Αν δεν έχουν, το αρχείο πιθανότατα θα είναι αδύνατο να επεξεργαστεί σωστά).

- Ένας χαρακτήρας (διπλό) εισαγωγικό σε ένα πεδίο πρέπει να εκπροσωπείται από δύο (διπλό) χαρακτήρες εισαγωγικά.

Η μορφή μπορεί να επεξεργάζεται από τα περισσότερα προγράμματα που ισχυρίζονται ότι διαβάζουν τα αρχεία CSV. Οι εξαιρέσεις είναι:

(α) τα προγράμματα ενδέχεται να μην υποστηρίζουν line-breaks, στο πλαίσιο quoted πεδίων, και
(β) τα προγράμματα μπορεί να μπερδέψουν την προαιρετική κεφαλίδα με τα δεδομένα ή να ερμηνεύσουν τα δεδομένα πρώτης γραμμής ως προαιρετική κεφαλίδα.

Βασικοί κανόνες

Πολλά ανεπίσημα έγγραφα υπάρχουν που περιγράφουν τις "CSV" μορφές.
Οι κανόνες αυτών και άλλων "CSV" προδιαγραφών και εφαρμογών έχουν ως εξής:

- Το CSV είναι μια οριοθετημένη μορφή δεδομένων που έχει πεδία/στήλες που διαχωρίζονται με τον χαρακτήρα κόμμα και εγγραφές/σειρές που τερματίζονται από νέες γραμμές.

- Ένα αρχείο CSV δεν απαιτεί μια ειδική κωδικοποίηση χαρακτήρων, σειρά byte, ή τερματισμού γραμμής μορφή (κάποια λογισμικά δεν υποστηρίζουν όλες τις line-end παραλλαγές).

- Μια εγγραφή τελειώνει σε έναν τερματισμό γραμμής. Ωστόσο, οι τερματιστές γραμμής μπορούν να ενσωματωθούν ως δεδομένα στο πεδίο, έτσι το λογισμικό

πρέπει να αναγνωρίσει τους quoted διαχωριστές-γραμμών, προκειμένου να συγκεντρώσει σωστά μια ολόκληρη εγγαρφή από τυχόν πολλαπλές γραμμές.

- Όλες οι εγγαρφές πρέπει να έχουν τον ίδιο αριθμό πεδίων, με την ίδια σειρά.

- Τα δεδομένα μέσα στα πεδία ερμηνεύονται ως μια ακολουθία χαρακτήρων, όχι ως μια ακολουθία από bits ή bytes. Για παράδειγμα, η αριθμητική ποσότητα 65.535 μπορεί να αναπαρασταθεί ως 5 χαρακτήρες ASCII "65535" (ή ίσως άλλες μορφές, όπως “0xFFFF", "000065535.000E + 00", κ.λπ.),αλλά όχι ως μια ακολουθία από 2 byte που προορίζονται να αντιμετωπίστούν ως ένας ενιαίος δυαδικός ακέραιος και όχι ως δύο χαρακτήρες.

- Γειτονικά πεδία πρέπει να χωρίζονται από ένα μόνο κόμμα. Ωστόσο, οι "CSV" μορφές ποικίλλουν σε μεγάλο βαθμό σε αυτή την επιλογή διαχωριστικών χαρακτήρων. Ειδικότερα, σε περιπτώσεις που το κόμμα χρησιμοποιείται ως διαχωριστικό δεκαδικών, τότε το ερωτηματικό, το TAB, ή άλλοι χαρακτήρες χρησιμοποιούνται αντ 'αυτού.

- Κάθε πεδίο μπορεί να είναι quoted (δηλαδή, να περικλείεται μέσα σε διπλά εισαγωγικά).

- Τα πεδία με ενσωματωμένα κόμματα ή διπλά εισαγωγικά πρέπει να είναι quoted.
  e.g. `1997,Ford,E350,"Super, luxurious truck"`

- Κάθε ένα από τα ενσωματωμένα διπλά εισαγωγικά θα πρέπει να αντιπροσωπεύεται από ένα ζεύγος διπλών εισαγωγικών.
  e.g. `1997,Ford,E350,"Super, ""luxurious"" truck"`

- Τα πεδία με ενσωματωμένη αλλαγή γραμμής πρέπει να είναι quoted (ωστόσο, πολλές εφαρμογές CSV δεν υποστηρίζουν τις ενσωματωμένες αλλαγές γραμμής).
  e.g.
```
1997,Ford,E350,"Go get one now
    they are going fast"
```

- Σε ορισμένες εφαρμογές CSV, τα πρώτα και τελευταία διαστήματα και tabs αγνονούνται. Τέτοιες περικοπές απαγορεύονται από το RFC 4180, το οποίο αναφέρει «τα διαστήματα θεωρούνται μέρος ενός πεδίου και δεν πρέπει να αγνοούνται."

- Σύμφωνα με το RFC 4180, τα διαστήματα έξω από εισαγωγικά σε ένα πεδίο δεν επιτρέπονται. Ωστόσο, το RFC αναφέρει επίσης ότι «τα διαστήματα θεωρούνται μέρος ενός πεδίου και δεν πρέπει να αγνοούνται" και "οι υλοποιητές “θα πρέπει να είναι συντηρητικοί σε ό, τι κάνεις, και ελεύθεροι σε ό, τι δέχεσαι από τους άλλους” (RFC 793), κατά την επεξεργασία των αρχείων CSV."

- Σε CSV εφαρμογές που έχουν περικόψει τα αρχικά ή τα τελικά κενά, τα πεδία με τέτοια διαστήματα ως ουσιαστικά δεδομένα πρέπει να είναι σε εισαγωγικά.
  e.g. 1997,Ford,E350, " Super luxurious truck "

- Η επεξεργασία διπλών εισαγωγικών χρειάζεται να ισχύσει μόνο αν το πεδίο αρχίζει με διπλό εισαγωγικό. Σημειώστε, ωστόσο, ότι διπλά εισαγωγικά δεν επιτρέπονται σε μη quoted πεδία, σύμφωνα με το RFC 4180.

- Η πρώτη εγγραφή μπορεί να είναι μια "κεφαλίδα", η οποία περιέχει ονόματα στηλών για καθένα από τα πεδία.

4

Βασικές λειτουργίες και σενάριο χρήσης της Εφαρμογής

4.1 Εισαγωγή

Στο κεφάλαιο αυτό θα περιγραφούν όλες οι διαφορετικές οθόνες της εφαρμογής, ο τρόπος σύνδεσης μεταξύ τους, οι κλάσεις στις οποίες υλοποιούνται, και όλες οι δυνατές επιλογές που έχει ο χρήστης όταν βρίσκεται σε συγκεκριμένη οθόνη. Τα διάφορα στιγμιότυπα της εφαρμογής παρουσιάζονται σε εικόνες και έπειτα εξηγούνται οι λειτουργίες τους, για να διευκολύνεται ο αναγνώστης στην κατανόηση των λειτουργιών αυτών.

Για διευκόλυνση της αναγνωσιμότητας του κειμένου στο κεφάλαιο αυτό θα παρουσιαστούν κάποια κομμάτια κώδικα με σημαντικές λειτουργίες της εφαρμογής, ενώ ολοκληρωμένα τα αρχεία αυτά παρατίθενται στο παράρτημα της παρούσας εργασίας. Παρατίθενται επομένως, τα κομμάτια του κώδικα για βασικές λειτουργίες της κάθε οθόνης, ενώ για πιο εξειδικευμένες λειτουργίες, προτείνεται στον αναγνώστη να μελετήσει τον ολοκληρωμένο κώδικα της αντίστοιχης κλάσης, που παρουσιάζεται στο παράρτημα.

4.2 Φόρτωση και αποθήκευση δεδομένων

Πριν ξεκινήσει η περιγραφή κάθε καρτέλας αναλυτικά, βλέπουμε τον τρόπο που γίνεται η φόρτωση ήδη υπάρχοντων δεδομένων, η αυτόματη δημιουργία του αρχείου "data1.xml" σε περίπτωση απουσίας του, καθώς και η αποθήκευση νέων στοιχείων. Όλες οι παραπάνω λειτουργίες βρίσκονται στην κλάση Database.cs και συγκεκριμένα υλοποιούνται με τις μεθόδους Load() και Save(). Ο σχετικός κώδικας φαίνεται παρακάτω:

```
public void Load()
    {
        if (System.IO.File.Exists(DataFile))
        {
            using (var stream = System.IO.File.OpenRead(DataFile))
            {
                var serializer = new XmlSerializer(typeof(Database));
                var obj = serializer.Deserialize(stream) as Database;

                this.Departments = obj.Departments ?? new List<Department>();
                this.Lessons = obj.Lessons ?? new List<Lesson>();
                this.Students = obj.Students ?? new List<Student>();
                this.Teachers = obj.Teachers ?? new List<Teacher>();
                this.GradeFactors = obj.GradeFactors?? new List<GradeDefinition>();

            }
        }
        else
        {
            this.Departments =  new List<Department>();
            this.Lessons =  new List<Lesson>();
            this.Students =new List<Student>();
            this.Teachers = new List<Teacher>();
            this.GradeFactors = new List<GradeDefinition>();
        }

    }
```

Η μέθοδος Load() ελέγχει αν υπάρχει το αρχείο "data1.xml", όπου καταχωρούνται όλα τα δεδομένα, και σε αυτή την περίπτωση διαβάζει από αυτό το αρχείο τα δεδομένα που υπάρχουν στις αντίστοιχες λίστες ή δημιουργεί καινούριες. Διαφορετικά, αν το αρχείο δεν υπάρχει καθόλου, δημιουργεί αυτόματα όλες τις λίστες που χρειάζονται.

Για την αποθήκευση νέων δεδομένων χρησιμοποιείται η μέθοδος Save(), η οποία όταν καλείται γράφει όλα τα νέα δεδομένα στο "data1.xml" αρχείο.

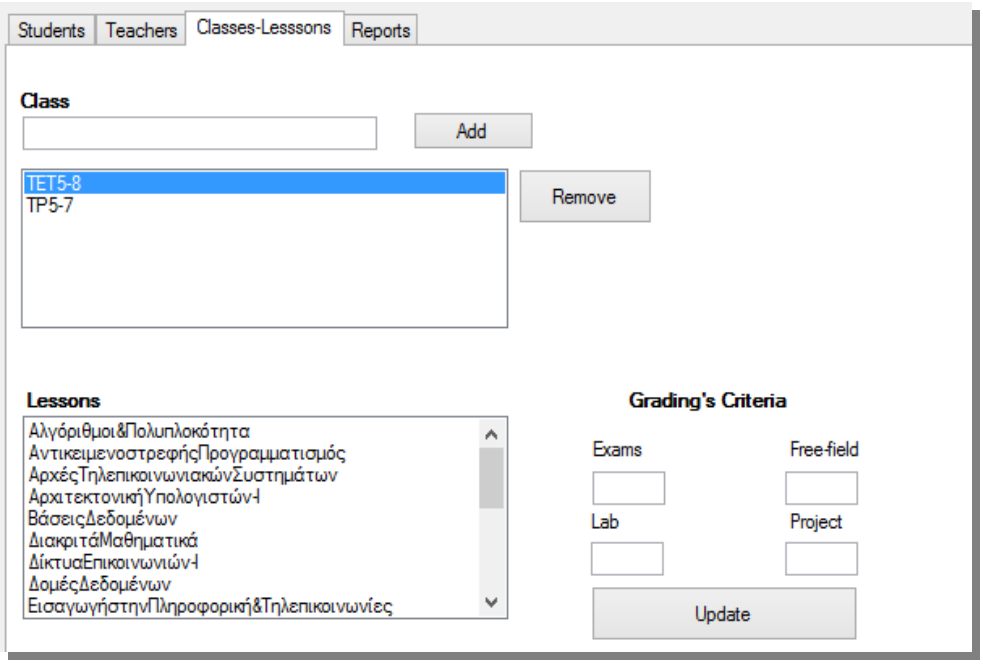
```
public void Save()
    {
        using (var writer = new System.IO.StreamWriter(DataFile))
        {
            var serializer = new XmlSerializer(this.GetType());
```

```
            serializer.Serialize(writer, this);
            writer.Flush();
        }
    }
```

4.3 Classes-Lessons tab

Η περιγραφή ξεκινάει από την τρίτη καρτέλα “Classes-Lessons” επειδή από εδώ είναι δόκιμο να ξεκινήσει η καταχώρηση στοιχείων, ώστε να είναι όλα έτοιμα για τις προηγούμενες καρτέλες. Όπως φανερώνει και το όνομά της στην καρτέλα αυτή μπορούμε να δημιουργήσουμε τις τάξεις της επιλογής μας καθώς και να διαχειριστούμε όποιο μάθημα μας ενδιαφέρει.

Η οθόνη της καρτέλας φαίνεται στην παρακάτω εικόνα και βλέπουμε αναλυτικότερα τις επιλογές της οθόνης αυτής:



Εικόνα 4.1- καρτέλα “Classes-Lessons”

Στο πεδίο Class εισάγουμε την τάξη(τμήμα) της επιλογής μας π.χ. ΤΕΤ5-8 και πατώντας το κουμπί Add προστίθεται στη λίστα των διαθέσιμων τάξεων. Είναι προτιμότερο να καταχωρούμε στοιχεία(τάξεις, μαθήματα, ονόματα) χωρίς να αφήνουμε κενά διαστήματα, αφού αυτό δημιουργεί πρόβλημα στο αρχείο excel που εξάγεται στο τέλος. Επιλέγοντας τώρα κάποια τάξη από τη λίστα μπορεί να διαγραφεί πατώντας το κουμπί Remove.

Στη λίστα Lessons εμφανίζονται τα μαθήματα που υπάρχουν στο αρχείο lessons.txt και ταξινομούνται με αλφαβητική σειρά.

```
        void LoadLessons()
    {
        Database db = new Database();
        db.Load();
        listLessons.Items.AddRange(db.Lessons.OrderBy(l => l.Name).ToArray());
        listLessons.DisplayMember = "Name";
    }
```

Το αρχείο “lessons.txt” έχει ήδη φορτωθεί και καταχωρηθεί στο “data1.xml” με τη βοήθεια της κλάσης LessonsLoader, όπως δείχνει το παρακάτω τμήμα κώδικα. Έτσι για ***αλλαγή των μαθημάτων δηλ. του lessons.txt*** αρχείου πρέπει να γίνει η αλλαγή του αρχείου ακόμα και μέσα από το περιβάλλον του Visual Studio να αποθηκευθεί το νέο αρχείο, και έπειτα στο φάκελο της εφαρμογής SchoolManagement>bin>Debug να ***διαγραφεί το αρχείο data1.xml***, ώστε στο επόμενο άνοιγμα της εφαρμογής να διαβαστεί το νέο αρχείο lessons.txt. Η ***διαγραφή του data1.xml*** βέβαια θα προκαλέσει και τη διαγραφή όλων των υπόλοιπων καταχωρήσεων (ονόματα, τάξεις) γι' αυτό πρέπει να είμαστε προσεκτικοί τη χρονική στιγμή που θα διαγράψουμε το data1.xml αρχείο.

```
class LessonsLoader
{
    public static bool LoadLessonsFromText()
    {
        Database db = new Database();
        db.Load();
        // check if the lessons are already loaded
        if (db.Lessons.Count > 0)
            return false;
        var lines = File.ReadLines("lessons.txt");
        int idseed = 1000;
        foreach (string line in lines)
        {
            db.Lessons.Add(new Lesson { Id = idseed, Name = line,Description =
  line });
            idseed++;
        }
        db.Save();
        return true;
    }
}
```

Στην ίδια οθόνη επιλέγοντας κάποιο μάθημα από τη λίστα μπορούμε δεξιά να ορίσουμε το ποσοστό βαρύτητας που έχει το κάθε κριτήριο αξιολόγησης. Αρκεί στο κάθε πεδίο να πληκτρολογήσουμε έναν ακέραιο αριθμό που θα εκφράζει το ποσοστό βαρύτητας του κάθε κριτηρίου, (π.χ. Πληκτρολογώντας στο Lab τον αριθμό 30 σημαίνει ότι ο βαθμός του εργαστηρίου έχει βαρύτητα 30% του τελικού βαθμού) που μετατρέπεται σε δεκαδικό αριθμό.

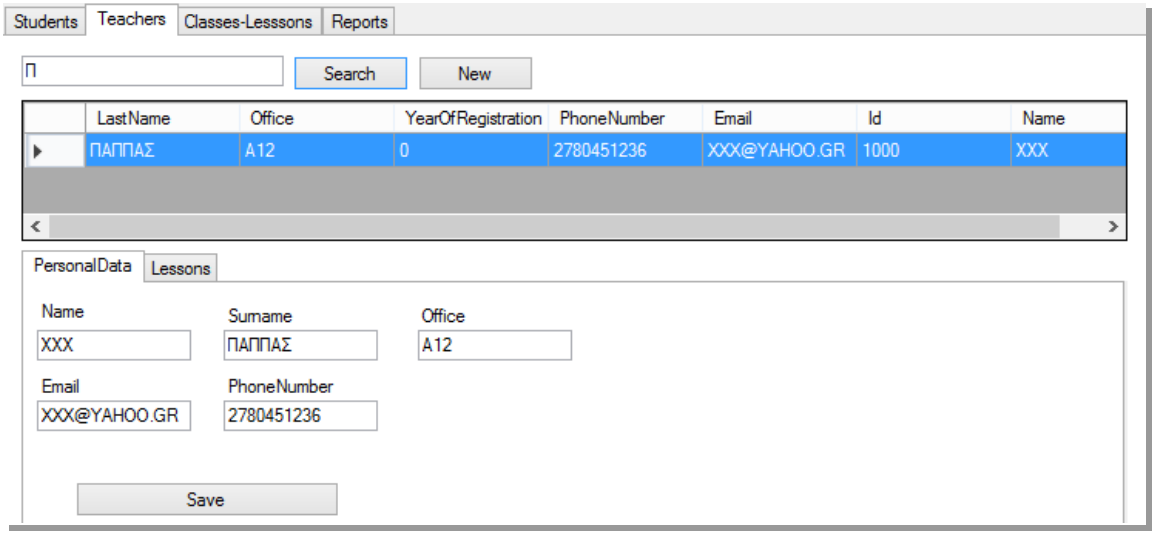
```
 private void btnAssign_Click(object sender, EventArgs e)
    {
        if (listLessons.SelectedIndex < 0)
            return;
        Database db = new Database();
```

```
        db.Load();
        var lesson = listLessons.SelectedItem as Lesson;
        var definition =db.GradeFactors.FirstOrDefault(g => g.ForLesson.Id ==
    lesson.Id);
        if (definition == null)
        {
            db.GradeFactors.Add(new GradeDefinition
            {
                ForLesson = lesson,
                FactorForExams = decimal.Parse(textExam.Text),
                FactorForLab = decimal.Parse(textLab.Text),
                FactorForOral = decimal.Parse(textVerbal.Text),
                FactorForProject = decimal.Parse(textProject.Text)
            });
        }
        else
        {
                definition.FactorForExams =decimal.Parse( textExam.Text );
                definition.FactorForLab = decimal.Parse(textLab.Text);
                definition.FactorForOral = decimal.Parse(textVerbal.Text);
                definition.FactorForProject = decimal.Parse(textProject.Text);
        }
        db.Save();
    }
```

Υπάρχει και ένα ελεύθερο πεδίο σε περίπτωση που υπάρξει κάποιος άλλος τρόπος εξέτασης (εκτός από τα γραπτά-εργαστήριο-εργασία), εάν δεν υπάρχει μπορούμε να εισάγουμε “0” στο πεδίο αυτό ή και σε οποιοδήποτε άλλο πεδίο που πιθανόν δεν συμμετέχει ως τρόπος εξέτασης και τότε στην εισαγωγή βαθμών δεν έχουμε δυνατότητα να καταχωρήσουμε βαθμό για το αντίστοιχο πεδίο. Με το πλήκτρο Update καταχωρούνται τα ποσοστά βαρύτητας και μπορούμε να προχωρήσουμε σε επόμενη καρτέλα.

4.4 Teachers tab

Εδώ θα περιγράψουμε την δεύτερη καρτέλα της εφαρμογής που αφορά τους καθηγητές και η αρχική της οθόνη φαίνεται παρακάτω. Μας δίνετε η δυνατότητα είτε να αναζητήσουμε ήδη καταχωρημένα στοιχεία - όπως φαίνεται στην εικόνα 4.2- γράφοντας μόνο ένα χαρακτήρα του ονόματος (έχει σημασία αν είναι μικρά ή κεφαλαία τα γράμματα- πρέπει η αναζήτηση να γίνει με όμοιο χαρακτήρα που έχει γίνει η καταχώρηση) και πατώντας το πλήκτρο Search, είτε να καταχωρήσουμε εκ' νέου έναν καθηγητή συμπληρώνοντας την καρτέλα *Personal Data*.

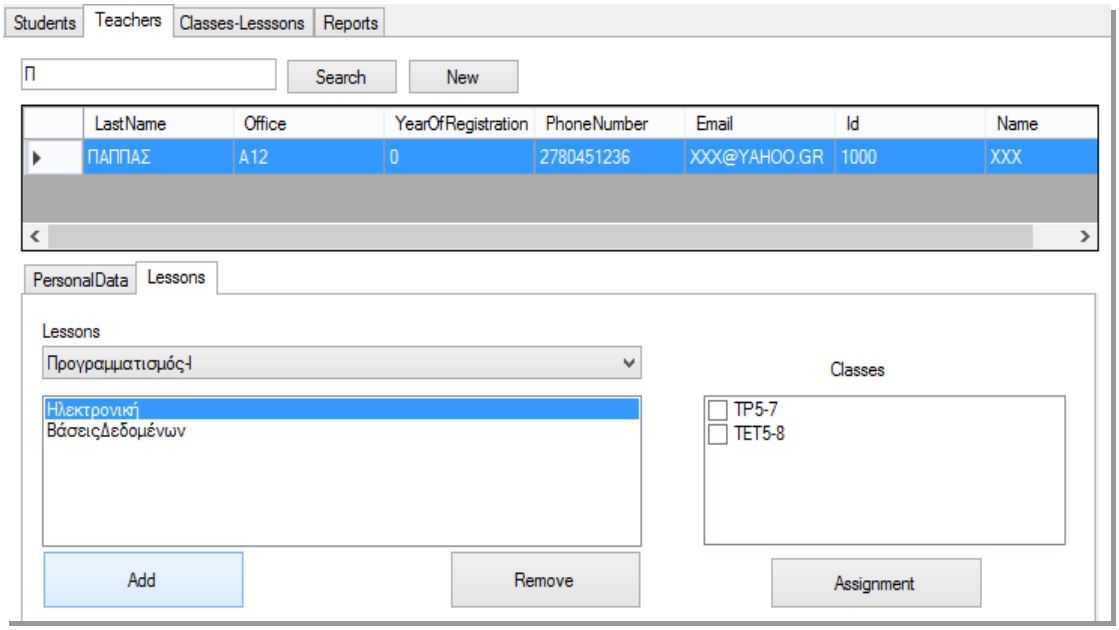


Εικόνα 4.2- καρτέλα "Teachers"- PeronalData tab

Αφού γίνει η καταχώρηση στοιχείων, πατάμε το πλήκτρο Save για αποθήκευση. Εάν δεν υπάρχει καμία σειρά στο dataGridView καλείται η μέθοδος AddTeacher() για καινούρια καταχώρηση, διαφορετικά καλείται η μέθοδος UpdateTeacher() η οποία ενημερώνει για τυχόν αλλαγές.

```
private void button1_Click(object sender, EventArgs e)
{
    if (dataGridView1.SelectedRows.Count < 1 || CreateNew)
        AddTeacher();
    else
    {
        UpdateTeacher();
    }
}
```

Στην ίδια καρτέλα – Teachers εκτός από τα Προσωπικά Στοιχεία ενός καθηγητή μπορούμε να διαχειριστούμε και τα μαθήματά του. Στο *tab Lessons* λοιπόν, επιλέγουμε το μάθημα που έχει αναλάβει ο καθηγητής ή και παραπάνω από ένα και με το πλήκτρο Add προστίθενται στη λίστα των μαθημάτων. Αντίστοιχα με το Remove μπορούμε να αφαιρέσουμε μαθήματα -εικόνα 4.3.



Εικόνα 4.3- καρτέλα "Teachers"- Lessons tab

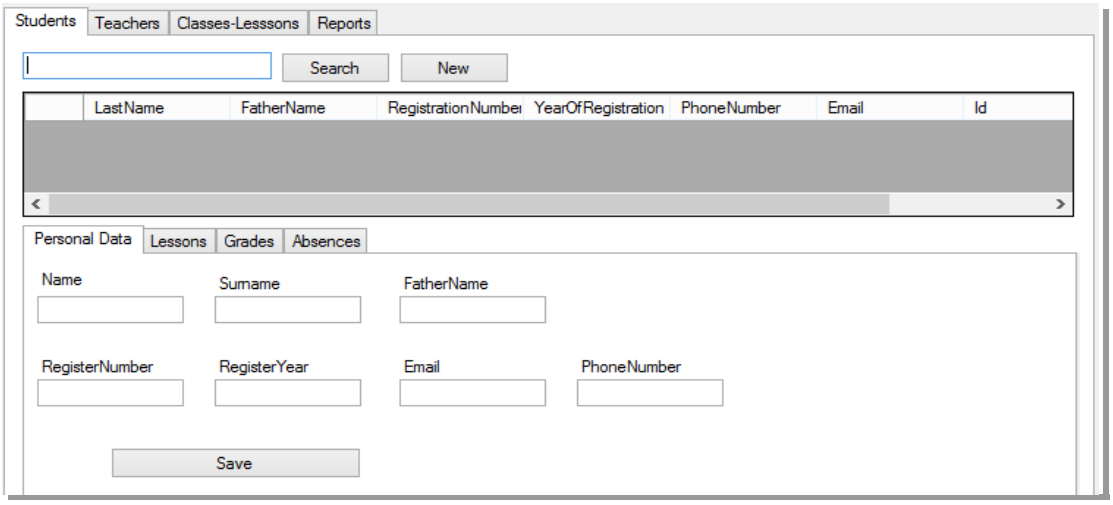
Δεξιότερα στην ίδια οθόνη παρατηρούμε ότι εμφανίζονται όλες οι διαθέσιμες τάξεις που έχουμε δημιουργήσει στην καρτέλα Classes-Lessons. Τώρα μπορούμε να επιλέξουμε την κατάλληλη τάξη(ή τάξεις) για το αντίστοιχο μάθημα και να την αναθέσουμε στον συγκεκριμένο καθηγητή που βρισκόμαστε, πατώντας το πλήκτρο Assignment. Έτσι για το κάθε μάθημα αναθέτουμε στον καθηγητή την αντίστοιχη τάξη (-εις) που έχει αναλάβει.

```
private void btnAssign_Click(object sender, EventArgs e)
   {
       Database db = new Database();
       db.Load();
       var gridTeacher = (Teacher)dataGridView1.SelectedRows[0].DataBoundItem;
       // load the teacher from database
       var teacher = db.Teachers.First(s => s.Id == gridTeacher.Id);
       // get the lesson from the list
       var lesson = listLessons.SelectedItem as Lesson;

       // iterate to all the departments and find the ones that are selected
       for (int i = 0; i < listDepartments.Items.Count; i++)
       {
           //get the department from the check box list
           var department = listDepartments.Items[i] as Department;

           if (listDepartments.GetItemChecked(i))
           {
               //  check if the lesson is already assigned to a department for this
               teacher
               if (!teacher.Assignements.Any(a => a.Department.Id == department.Id &&
               a.Lesson.Id == lesson.Id))
               {
                   // assign them to the teacher
                   teacher.Assignements.Add(new LessonAssignement { Department =
                  department, Lesson = lesson });
```

```
                }
            }
            else
                teacher.Assignements.RemoveAll(p=>p.Lesson.Id == lesson.Id &&
                 department.Id == p.Department.Id);
        }
        // save the changes
        db.Save();
    }
```

4.5 Students tab

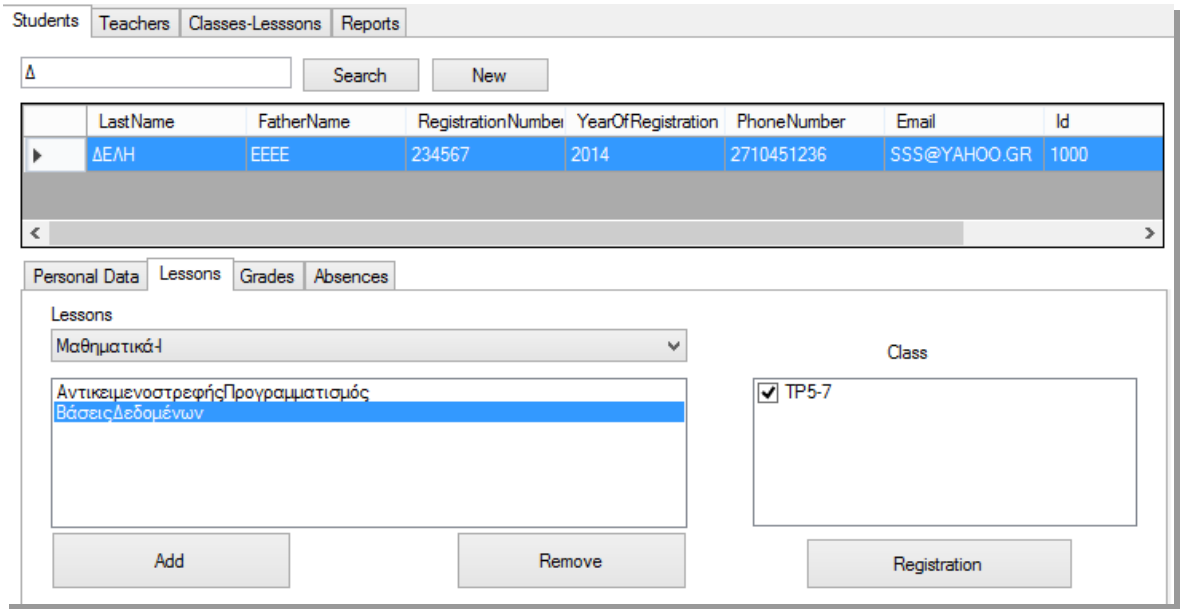
Η πρώτη καρτέλα (Students *tab*) της εφαρμογής αναφέρεται στους φοιτητές και θα δούμε αναλυτικά όλα τα στοιχεία που μπορούμε να διαχειριστούμε σχετικά με τους φοιτητές όσον αφορά, τα προσωπικά τους δεδομένα, τα μαθήματά τους, τους βαθμούς και τις απουσίες τους. Η αρχική οθόνη της καρτέλας (και συγκεκριμένα το *tab Personal Data*)φαίνεται στην εικόνα 4.4.



Εικόνα 4.4- καρτέλα "Students"- Personal Data tab

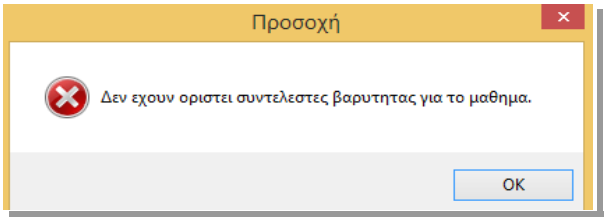
Αντίστοιχα με τους καθηγητές, εδώ μπορούμε να αναζητήσουμε κάποιες υπάρχουσες εγγραφές πληκτρολογώντας το πρώτο γράμμα του επωνύμου τους (μικρό ή κεφαλαίο -ανάλογα πως έχουν γίνει οι καταχωρήσεις), ή να καταχωρήσουμε μια καινούρια εγγραφή και να την αποθηκεύσουμε με το Save.

Στο *tab Lessons* (εικόνα 4.5) μπορούμε να εγγράψουμε τους φοιτητές στα συγκεκριμένα μαθήματα και τάξεις που παρακολουθούν. Επιλέγουμε από τη λίστα Lessons τα μαθήματα που παρακολουθεί ο φοιτητής και με το κουμπί Add τα προσθέτουμε στη λίστα των μαθημάτων του. Πατώντας πάνω στο κάθε μάθημα παρατηρούμε οτι δεξιά στη λίστα Class εμφανίζονται μόνο οι τάξεις που έχει αναλάβει κάποιος καθηγητής και όχι όλες οι διαθέσιμες τάξεις που τυχόν έχουμε δημιουργήσει στη καρτέλα Classes-Lessons.



Εικόνα 4.5- καρτέλα "Students"- Lessons tab

Έτσι επιλέγοντας και την κατάλληλη τάξη για το κάθε μάθημα και πατώντας το κουμπί Registration (όπως φαίνεται και στο παρακάτω τμήμα κώδικα) ο φοιτητής εγγράφεται στη συγκεκριμένη τάξη του μαθήματος, που σίγουρα έχει αναλάβει κάποιος καθηγητής.

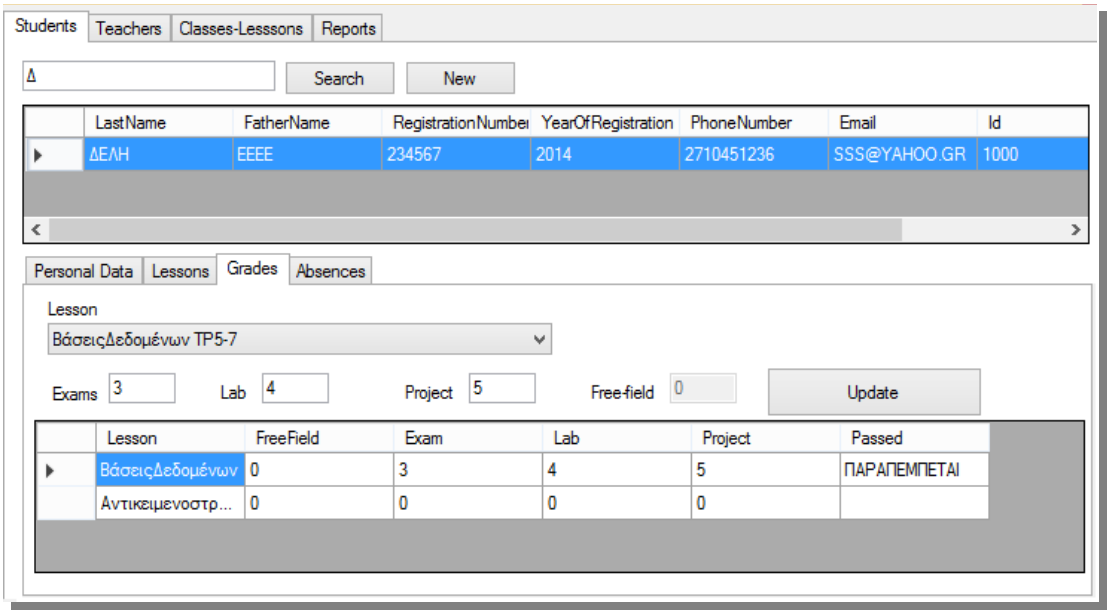



```
private void btnAssign_Click(object sender, EventArgs e)
   {
       Database db = new Database();
       db.Load();
       var gridStudent = (Student)dataGridView1.SelectedRows[0].DataBoundItem;
       // load the student from database
       var student = db.Students.First(s => s.Id == gridStudent.Id);
       // get the lesson from the list
       var lesson = listLessons.SelectedItem as Lesson;
       //find the registration of the student for this lesson
       var registration = student.RegisteredLessons.Find(l => l.Lesson.Id ==
         lesson.Id);
       // iterate to all the departments and find the ones that are selected
       for (int i = 0; i < listDepartments.Items.Count; i++)
       {
           //get the department from the check box list
           var department = listDepartments.Items[i] as Department;
```

```
            //if it is selected
            if (listDepartments.GetItemChecked(i))
            {
                //update the registration with the selected department
                registration.Department = department;
                break;
            }
        }
        // save the changes
        db.Save();
    }
```

Η επόμενη καρτέλα που αφορά τους φοιτητές είναι το *Grades tab* και μπορούμε να εισάγουμε την βαθμολογία τους. Στη λίστα Lesson επιλέγουμε το μάθημα που μας ενδιαφέρει και αυτομάτως ελέγχεται αν έχουν οριστεί συντελεστές βαρύτητας (καρτέλα Classes-Lessons) για το μάθημα αυτό. Εάν όχι τότε το κατάλληλο μήνυμα (όπως φαίνεται δίπλα) μας ενημερώνει και δεν μπορούμε να προχωρήσουμε στην καταχώρηση βαθμολογίας.

Διαφορετικά, μπορούμε να καταχωρήσουμε κανονικά την βαθμολογία για κάθε κριτήριο βαθμολόγησης (γραπτά, εργασία, κ.λ.π.). Μόνο αν κάποιος συντελεστής βαρύτητας έχει οριστεί σε μηδέν "0", τότε το αντίστοιχο πεδίο εμφανίζεται ανενεργό (εικόνα 4.6 -FreeField) και εκεί δεν επιτρέπεται η εισαγωγή βαθμού.



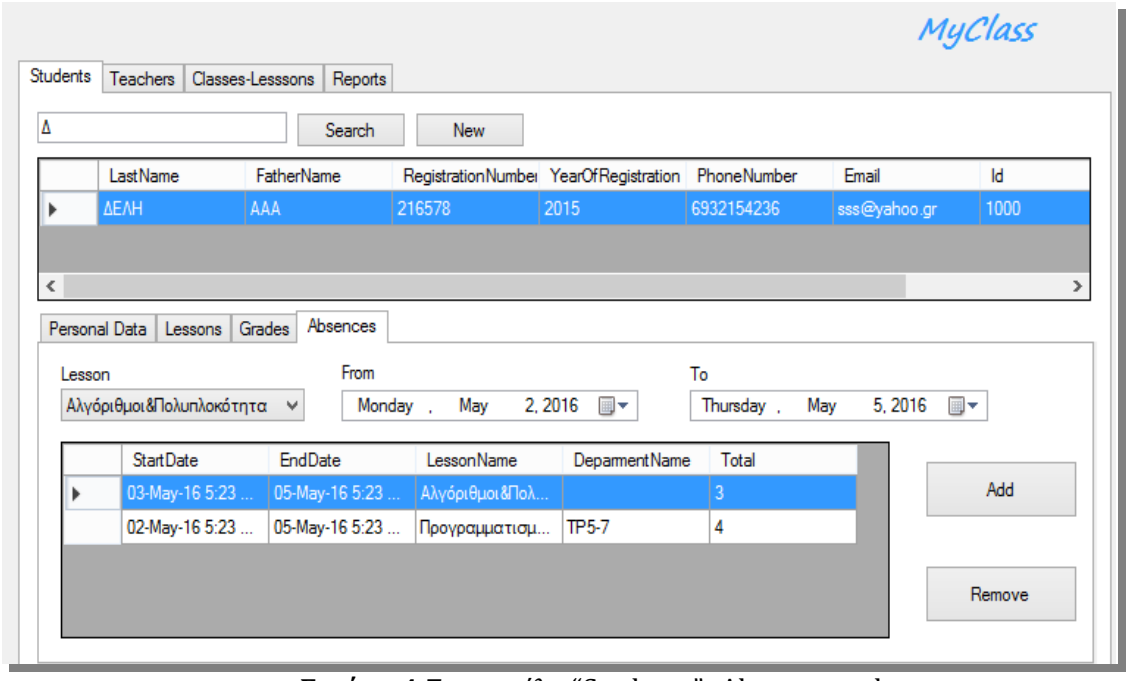
Εικόνα 4.6- καρτέλα "Students"- Grades tab

Πατώντας το κουμπί Update αποθηκεύονται οι βαθμοί σε δεκαδική μορφή όπως φαίνεται και στον παρακάτω κώδικα, για να μπορέσουμε να τους χρησιμοποιήσουμε στον υπολογισμό του αποτελέσματος.

```
private void btnSetGrade_Click(object sender, EventArgs e)
```

```
{
    Database db = new Database();
    db.Load();
    var gridStudent = (Student)dataGridView1.SelectedRows[0].DataBoundItem;

    var dbstudent = db.Students.First(dbs => dbs.Id == gridStudent.Id);
    var lesson = cboLessonForGrades.SelectedItem as LessonAssignement;
    var grade = dbstudent.Grades.Where(g => g.ForLesson.Lesson.Id ==
    lesson.Lesson.Id).First();

            grade.Value = decimal.Parse(textExam.Text);
            grade.ValueForLab = decimal.Parse(textLab.Text);
            grade.ValueForOral = decimal.Parse(textVerbal.Text);
            grade.ValueForProject = decimal.Parse(textProject.Text);

    if (dbstudent.Grades.Count > 0)
    {
        dataGridGrades.DataSource = dbstudent.Grades.ToFlat();
    }
    db.Save();
}
```

Η τελευταία καρτέλα που αφορά τους φοιτητές είναι το *tab Absences* και έχει να κάνει με την καταχώρηση απουσιών.



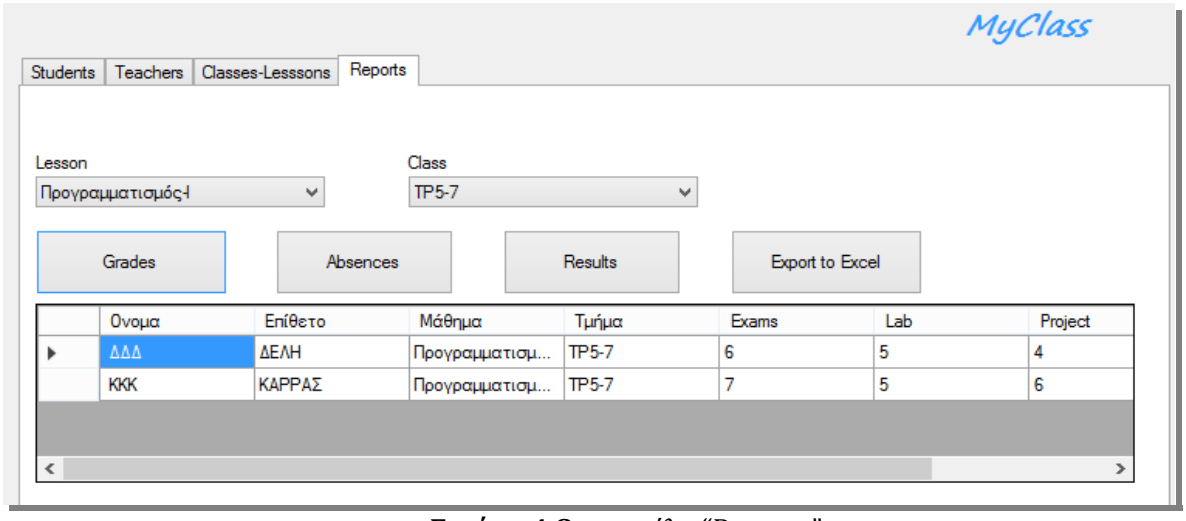
Εικόνα 4.7- καρτέλα "Students"- Absences tab

Στη λίστα Lesson επιλέγουμε το μάθημα που μας ενδιαφέρει και έπειτα απο το From πατώντας το βελάκι εμφανίζεται ένα ημερολόγιο που επιλέγουμε την ημερομηνία έναρξης απουσιών, ενώ απο το To επιλέγουμε την ημερομηνία λήξης απουσιών. Η καταχώρηση γίνεται πατώντας το πλήκτρο Add. Όταν η ημερομηνία έναρξης και λήξης απουσιών είναι ίδια, καταχωρείται μία μόνο απουσία. Το σύνολο των απουσιών για συγκεκριμένες ημερομηνίες φαίνεται στη στήλη Total, και αν

επιθυμούμε την διαγραφή κάποιων απουσιών επιλέγουμε την αντίστοιχη γραμμή και παταμε το πλήκτρο Remove.

4.6 Reports tab

Η αρχική οθόνη της τελευταίας καρτέλας φαίνεται στην εικόνα 4.8 και από αυτό το σημείο μπορούμε να εμφανίσουμε σε αρχείο Excel τα δεδομένα μας, που αφορούν βαθμολογίες, απουσίες και αποτελέσματα φοιτητών. Πριν προχωρήσουμε σε καθεμία από αυτές τις επιλογές να δούμε τον τρόπο με τον οποίο δημιουργείται το Csv αρχείο και ανοίγει σε μορφή Excel.



Εικόνα 4.8- καρτέλα "Reports"

Η μέθοδος που χρησιμοποιείται είναι ToCsV(), η οποία αρχικά κάνει εξαγωγή των τίτλων του DataGridView και έπειτα των δεδομένων του κάθε κελιού και της κάθε γραμμής.

```
private void ToCsV(DataGridView dGV, string filename)
  {
      string stOutput = " ";
      // Export titles:
      string sHeaders = " ";

      for (int j = 0; j < dGV.Columns.Count; j++)
      sHeaders = sHeaders.ToString() + Convert.ToString(dGV.Columns[j].HeaderText)+
        "\t";
      stOutput += sHeaders + "\r\n";
      // Export data.
      for (int i = 0; i < dGV.RowCount; i++)
      {
          string stLine = " ";
          for (int j = 0; j < dGV.Rows[i].Cells.Count; j++)
              stLine = stLine.ToString() +
```

```
                Convert.ToString(dGV.Rows[i].Cells[j].Value) + "\t";
            stOutput += stLine + "\r\n";
        }
        Encoding utf16 = Encoding.Default;// GetEncoding(1253);
        byte[] output = utf16.GetBytes(stOutput);
        FileStream fs = new FileStream(filename, FileMode.Create);
        BinaryWriter bw = new BinaryWriter(fs);
        bw.Write(output, 0, output.Length); //write the encoded file
        bw.Flush();
        bw.Close();
        fs.Close();
    }
```

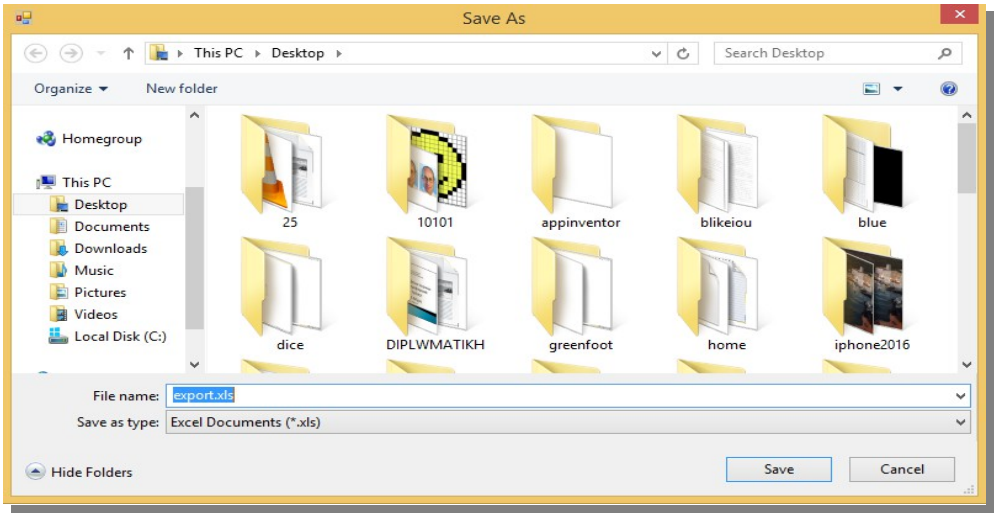
Έπειτα με τη βοήθεια της κλάσης Encoding του System.Text namespace και συγκεκριμένα της Default ιδιότητας η οποία παίρνει μια κωδικοποίηση για την τρέχουσα σελίδα κώδικα ANSI του λειτουργικού μας συστήματος, γίνεται η κωδικοποίηση. Στη συνέχεια δημιουργούμε ένα byte array που θα περιέχει το αποτέλεσμα της κωδικοποιήσης των χαρακτήρων του συγκεκριμένου string (stOutput).

Χρησιμοποιώντας την FileStream κλάση αρχικοποιούμε ένα στιγμιότυπο για το συγκεκριμένο path και τη λειτουργία δημιουργίας ενός νέου αρχείου. Τέλος, απο την κλάση BinaryWriter αρχικοποιούμε ένα νέο στιγμιότυπο για το συγκεκριμένο stream που δημιουργήσαμε και χρησιμοποιούμε τη μεθόδο Write(), η οποία γράφει το κωδικοποιημένο αρχείο. Με τον τρόπο αυτό, τα κωδικοποιημένα δεδομένα έχουν αποθηκευθεί σε μια πινακοειδή μορφή, που μπορεί εύκολα να εισαχθεί σε υπολογιστικά φύλλα.

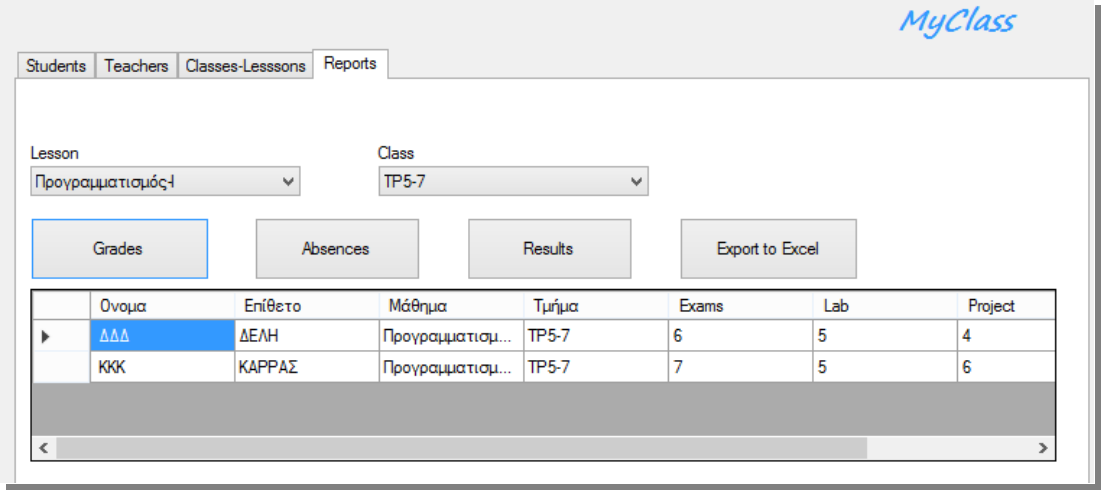
Όταν το πλήκτρο της εφαρμογής “Export to Excel” πατηθεί, με τη βοήθεια της κλάσης SaveFileDialog, ορίζουμε οτι ο τύπος του αρχείου στο παράθυρο διαλόγου θα είναι Excel Documents (*.xls) καθώς και το όνομα του αρχείου.

```
private void button1_Click(object sender, EventArgs e)
 {
     SaveFileDialog sfd = new SaveFileDialog();
     sfd.Filter = "Excel Documents (*.xls)|*.xls";
     sfd.FileName = "export.xls";
     if (sfd.ShowDialog() == DialogResult.OK)
     {
         ToCsV(dataGridView1, sfd.FileName);
     }
 }
```

Ο χρήστης βρίσκεται μπροστά σε αυτό το παράθυρο διαλόγου και όταν επιλέξει Save/OK, γίνεται η αποθήκευση του αρχείου σε μορφή .xls και έπειτα μπορούμε να διαχειριστούμε το αρχείο μας με όλες τις δυνατότητες των υπολογιστικών φύλλων.



Αφού περιγράψαμε τον τρόπο με τον οποίο δημιουργείται και αποθηκεύεται το αρχείο excel, επιστρέφουμε στην εφαρμογή και στη Reports tab να δούμε τις επιλογές που υπάρχουν στην καρτέλα αυτή.



Εικόνα 4.9- καρτέλα "Reports"

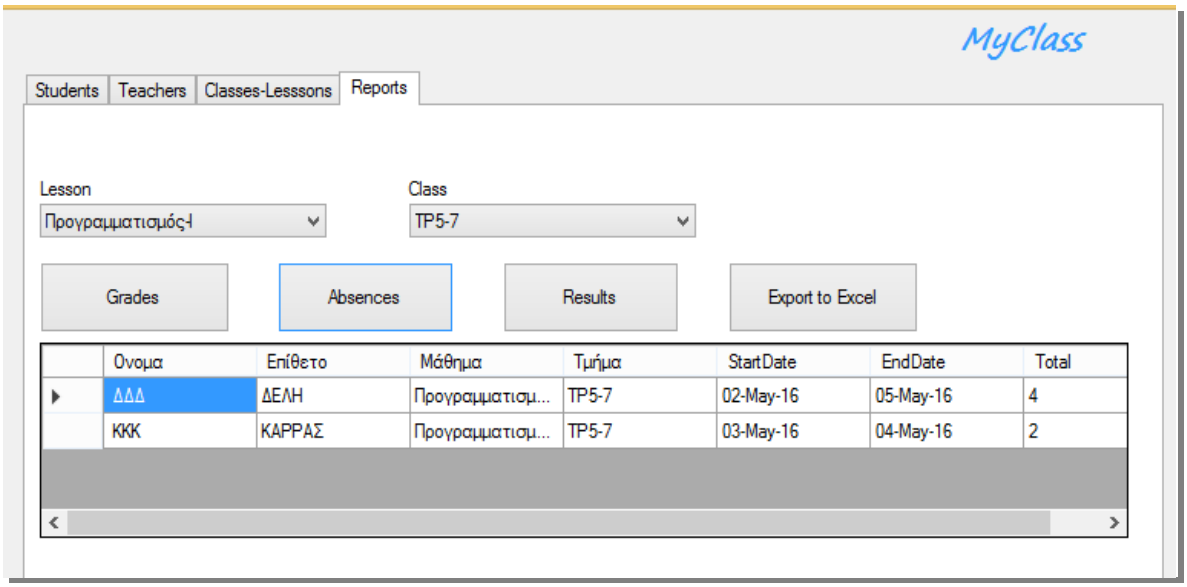
Αρχικά πρέπει να επιλέξουμε το μάθημα και την τάξη για την οποία επιθυμούμε να δούμε στοιχεία. Το πρώτο κουμπί, όπως φαίνεται και στην εικόνα 4.9 είναι το κουμπί Grades και αφορά την βαθμολογία. Με το αντίστοιχο ερώτημα, με τη βοήθεια των LINQ- queries (έχουν περιγραφεί στο 2ο Κεφάλαιο) , επιλέγουμε τα δεδομένα και τις επικεφαλίδες που θα εμφανιστούν στο dataGridView1 και έτσι αντλούμε τις βαθμολογίες για όλους τους τρόπους εξέτασης, όλων των φοιτητών που ανήκουν στο συγκεκριμένο μάθημα και τμήμα.

```
private void btnGrades_Click(object sender, EventArgs e)
    {
        Database db = new Database();
        db.Load();
        var lesson = cboLessons.SelectedItem.ToString();

        var department = cboDepartment.SelectedItem as Department;


        var gradesQuery =   from student in db.Students
                            where student.Grades.Any(g => g.ForLesson.Lesson.Name ==
                              lesson && g.ForLesson.Department.Name == department.Name)
                            let grade =
  student.Grades.First(g=>g.ForLesson.Lesson.Name== lesson &&
  g.ForLesson.Department.Name == department.Name)
                            select new { Name =" "+ student.Name+" ",
                                         LastName =student.LastName+" ",
```

```
                                    Lesson =grade.ForLesson.Lesson.Name+" ",
                                Department =grade.ForLesson.Department.Name+" ",
                                 Value=grade.Value.ToString()+" " ,
                                 ValueForLab=grade.ValueForLab.ToString()+" " ,
                                ValueForProject=grade.ValueForProject.ToString()
                                    +" ",
                                ValueForOral=grade.ValueForOral.ToString()+" "};

            // the data from the query
            dataGridView1.DataSource = gradesQuery.ToList();


             // change header names
             dataGridView1.Columns[0].HeaderText = "Ονομα ";
             dataGridView1.Columns[1].HeaderText = "Επίθετο ";
             dataGridView1.Columns[2].HeaderText = "Μάθημα ";
             dataGridView1.Columns[3].HeaderText = "Τμήμα ";
             dataGridView1.Columns[4].HeaderText = "Exams ";
             dataGridView1.Columns[5].HeaderText = "Lab ";
             dataGridView1.Columns[6].HeaderText = "Project ";
             dataGridView1.Columns[7].HeaderText = "Free-field ";

        }
```

Προσοχή: τα κενά διαστήματα που διαβάζονται δίπλα από τις τιμές είναι σημαντικά για να καταχωρούνται τα δεδομένα μας στη σωστή στήλη και κατά συνέπεια να μην υπάρχει πρόβλημα και στις στήλες του αρχείου excel που παράγεται αργότερα.

Η επόμενη δυνατότητα που υπάρχει στην ίδια καρτέλα είναι με το κουμπί Absences, όπως φαίνεται στην εικόνα 4.10 και αφορά τις απουσίες των φοιτητών.



Εικόνα 4.10- καρτέλα "Reports"

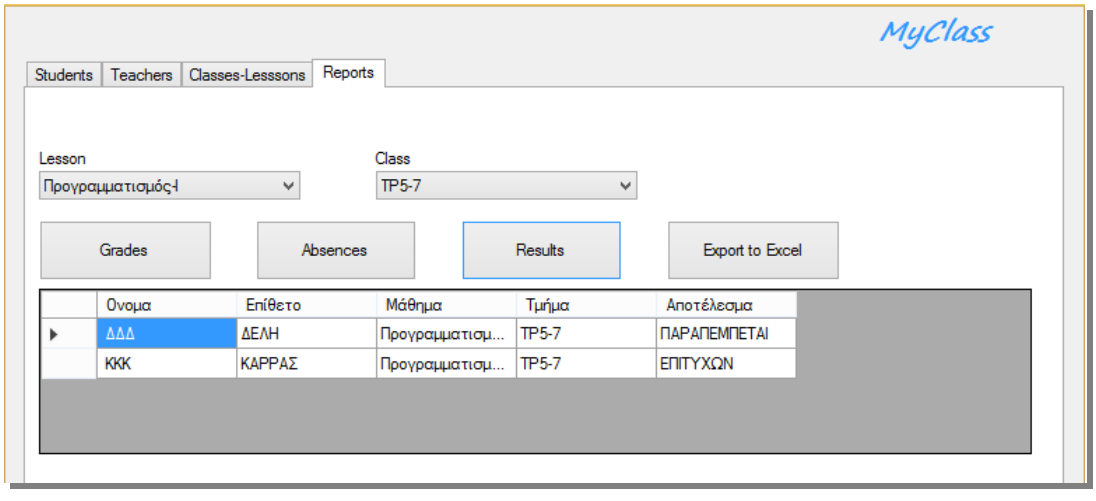
Πάλι με το κατάλληλο ερώτημα, όπως φαίνεται στο παρακάτω τμήμα κώδικα, μπορούμε να εμφανίσουμε τις ημερομηνίες καθώς και το σύνολο των απουσιών των

φοιτητών που παρακολουθούν το συγκεκριμένο μάθημα και τάξη.

```
 private void btnAbsences_Click(object sender, EventArgs e)
        {
            Database db = new Database();
            db.Load();
            var lesson = cboLessons.SelectedItem.ToString();

            var department = cboDepartment.SelectedItem as Department;

            var absQuery = from student in db.Students
                           from a in student.Absences
                           where a.RegisteredLesson.Lesson.Name == lesson &&
a.RegisteredLesson.Department.Name == department.Name
                           select new
                           {
                               Name = " " + student.Name + " ",
                               LastName = student.LastName + " ",
                               Lesson = a.RegisteredLesson.Lesson.Name + " ",
                               Department = a.RegisteredLesson.Department.Name + " ",
                               StartDate = a.StartDate.ToString() + " ",
                               EndDate = a.EndDate.ToString() + " ",
                               Total = a.EndDate.Subtract(a.StartDate).Days + 1
                           };
                    dataGridView1.DataSource = absQuery.ToList();

            // change header names
            dataGridView1.Columns[0].HeaderText = "Ονομα ";
            dataGridView1.Columns[1].HeaderText = "Επίθετο ";
            dataGridView1.Columns[2].HeaderText = "Μάθημα ";
            dataGridView1.Columns[3].HeaderText = "Τμήμα ";
            dataGridView1.Columns[4].HeaderText = "StartDate ";
            dataGridView1.Columns[5].HeaderText = "EndDate ";
            dataGridView1.Columns[6].HeaderText = "Total  ";
        }
```

Έτσι έχουμε τη δυνατότητα να έχουμε μια ολοκληρωμένη εικόνα για τον κάθε φοιτητή εύκολα, ώστε να κρίνουμε αν η φοιτησή του είναι επαρκής απο άποψη απουσιών ή όχι.

Η τελευταία επιλογή είναι το κουμπί Results και αφορά τα αποτελέσματα της συγκεκριμένης τάξης.



Εικόνα 4.11- καρτέλα "Reports"

Με το αντίστοιχο ερώτημα, εμφανίζουμε αν οι φοιτητές είναι επιτυχόντες ή αποτυχόντες στο συγκεκριμένο μάθημα. Το αποτέλεσμα προκύπτει, σύμφωνα με τη βαρύτητα των κριτηρίων αξιολόγησης που έχουμε ορίσει στην καρτέλα Classes-Lessons, αν όλα τα κριτήρια ανάλογα με τη βαρύτητά τους συγκεντρώνουν βαθμό μεγαλύτερο του 5 ο χαρακτηρισμός είναι "επιτυχών" διαφορετικά "παραπέμπεται".

```
private void btnResults_Click(object sender, EventArgs e)
{
    Database db = new Database();
    db.Load();
    var lesson = cboLessons.SelectedItem.ToString();

    var department = cboDepartment.SelectedItem as Department;

    var resultQuery = from student in db.Students
                      where student.Grades.Any(g => g.ForLesson.Lesson.Name ==
                          lesson && g.ForLesson.Department.Name == department.Name)
                      let grade = student.Grades.First(g => g.ForLesson.Lesson.Name
                          == lesson)
                      select new
                      {
                          Name = " "+ student.Name+" ",
                          LastName = student.LastName+" ",
                          Lesson = grade.ForLesson.Lesson.Name + " ",
                          Department = grade.ForLesson.Department.Name + " ",
                          Passed = CalculateResult(grade)
                      };


    // the data from the query
    dataGridView1.DataSource = resultQuery.ToList();



    // change header names
    dataGridView1.Columns[0].HeaderText = "Ονομα ";
    dataGridView1.Columns[1].HeaderText = "Επίθετο ";
    dataGridView1.Columns[2].HeaderText = "Μάθημα ";
    dataGridView1.Columns[3].HeaderText = "Τμήμα ";
    dataGridView1.Columns[4].HeaderText = "Αποτέλεσμα ";
}
```

Η συγκεκριμένη επιλογή είναι ιδιαίτερα χρήσιμη για την έκδοση τελικών αποτελεσμάτων του κάθε μαθήματος.

5

Συμπεράσματα και μελλοντικές εξελίξεις

5.1 Περίληψη και συμπεράσματα

Στην παρούσα εργασία ασχοληθήκαμε με την ανάπτυξη μιας εφαρμογής, που επιτρέπει στους χρήστες να οργανώσουν και να διαχειριστούν εύκολα μια ηλεκτρονική τάξη. Με τη βοήθεια του προγραμματιστικού περιβάλλοντος Visual Studio, η εφαρμογή μας δίνει τη δυνατότητα για καταχώρηση προσωπικών στοιχείων μαθητών και καθηγητών, βαθμολογιών, απουσιών και εξαγωγή όλων αυτών σε αρχείο excel.

Το Visual Studio είναι ένα περιβάλλον που παρέχει όλα τα σύγχρονα εργαλεία για την διαχείριση και ανάπτυξη έξυπνων, μοντέρνων εφαρμογών που καλύπτουν ανάγκες απαιτητικών χρηστών αλλά και προγραμματιστών. Μέσα απο το συγκεκριμένο περιβάλλον μπορούν να σχεδιαστούν από εφαρμογές κονσόλας, παραθυρικές εφαρμογές μέχρι παιχνίδια, εμπορικές ή web εφαρμογές ανάλογα με το αντικείμενο ενδιαφέροντος του κάθε χρήστη.

Η συγκεκριμένη εφαρμογή που δημιουργήσαμε κρίνεται σχετικά εύκολη στη χρήση της, αφού δεν απαιτεί ιδιαίτερες γνώσεις του χρήστη και ο καθένας μπορεί να την χειριστεί με μια μικρή εξοικείωση μαζί της. Όσο για τη δημιουργία της εφαρμογής απαιτούνται ιδιαίτερες γνώσεις της γλώσσας προγραμματισμού C# και αρκετή εξοικείωση με τις λειτουργίες και τις δυνατότητες που προσφέρει η γλώσσα. Η έρευνα που πραγματοποιήθηκε για την ολοκλήρωση της εφαρμογής ήταν ιδιαίτερα

ενδιαφέρουσα, από το σχεδιαστικό κομμάτι μέχρι την κατανόηση δύσκολων εννοιών των κλάσεων, μεθόδων και ιδιοτήτων που χρησιμοποιήθηκαν για την επίτευξη του επιθυμητού στόχου. Η εφαρμογή μπορεί να εκτελείται σε οποιοδήποτε υπολογιστή desktop ή laptop που διαθέτει εγκατεστημένο το περιβάλλον του Visual Studio και δεν παρουσιάζει ιδιαίτερες ασυμβατότητες με την έκδοση 2013.

5.2 **Μελλοντικές εξελίξεις**

Το Visual Studio ως περιβάλλον ανάπτυξης και η C# ως γλώσσα προγραμματισμού δίνουν πολλές δυνατότητες στους προγραμματιστές και η συγκεκριμένη εφαρμογή είναι εύκολα επεκτάσιμη. Μπορούν να προστεθούν επιπλέον λειτουργίες τόσο για απαιτήσεις του καθηγητή που διαχειρίζεται την εφαρμογή, όσο και απο χειρισμό της εφαρμογής από τη γραμματεία ενός τμήματος.

Κάποιες πιθανές προτάσεις είναι οι παρακάτω:

- καταχώρηση ωρών διδασκαλίας του εκάστοτε καθηγητή για υπολογισμό μισθοδοσίας ή κάποιο άλλο σκοπό
- δυνατότητα εισαγωγής στοιχείων των νέων φοιτητών απευθείας από τη γραμματεία, από κάποια αρχείο excel ώστε να είναι έτοιμοι προς επεξεργασία από την καθηγητή
- εξαγωγή τελικών αποτελεσμάτων σύμφωνα με τον αριθμό απουσιών ή σύμφωνα με την επίδοση σε συγκεκριμένο κριτήριο βαθμολόγησης
- εκτύπωση βεβαιώσεων παρακολούθησης για συγκεκριμένα μαθήματα βάση των τελικών αποτελεσμάτων

ПАРАРТНМА

Εδώ παρατίθεται αναλυτικά ο κώδικας της εφαρμογής. Η εφαρμογή αποτελείται από 2 projects, το SchoolManagement και το SchoolManagement.Data και παρουσιάζονται όλες οι κλάσεις ταξινομημένες αλφαβητικά, ξεκινώντας απο το project SchoolManagement.Data που δημιουργήθηκε πρώτα.

SchoolManagement.Data (project)

Absence.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SchoolManagement.Data
{
    public class Absence
    {
        public DateTime StartDate { get; set; }
        public DateTime EndDate { get; set; }

        public LessonAssignement RegisteredLesson { get; set; }
    }
}
```

BaseObject.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SchoolManagement.Data
{
    public class BaseObject
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

Database.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace SchoolManagement.Data
{
    public class Database
    {
```

```
        private const string DataFile = "data1.xml";
        public List<Teacher> Teachers { get; set; }
        public List<Lesson> Lessons { get; set; }
        public List<Student> Students { get; set; }
        public List<Department> Departments { get; set; }

        public List<GradeDefinition> GradeFactors { get; set; }
        public void Load()
        {
            if (System.IO.File.Exists(DataFile))
            {
                using (var stream = System.IO.File.OpenRead(DataFile))
                {
                    var serializer = new XmlSerializer(typeof(Database));
                    var obj = serializer.Deserialize(stream) as Database;

                    this.Departments = obj.Departments ?? new List<Department>();
                    this.Lessons = obj.Lessons ?? new List<Lesson>();
                    this.Students = obj.Students ?? new List<Student>();
                    this.Teachers = obj.Teachers ?? new List<Teacher>();
                    this.GradeFactors = obj.GradeFactors?? new List<GradeDefinition>();

                }
            }
            else
            {
                this.Departments =  new List<Department>();
                this.Lessons =  new List<Lesson>();
                this.Students =new List<Student>();
                this.Teachers = new List<Teacher>();
                this.GradeFactors = new List<GradeDefinition>();
            }

        }
        public void Save()
        {
            using (var writer = new System.IO.StreamWriter(DataFile))
            {
                var serializer = new XmlSerializer(this.GetType());
                serializer.Serialize(writer, this);
                writer.Flush();
            }
        }
    }
}
```

Department.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SchoolManagement.Data
{
    public class Department :BaseObject
    {
    }
}
```

Grade.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SchoolManagement.Data
{
    public class Grade
    {
        public decimal Value { get; set; }

        public LessonAssignement ForLesson { get; set; }

        public decimal ValueForOral { get; set; }

        public decimal ValueForProject { get; set; }

        public decimal ValueForLab { get; set; }
    }
}
```

GradeDefinition.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SchoolManagement.Data
{
    public class GradeDefinition
    {

        public Lesson ForLesson { get; set; }

        public decimal FactorForExams { get; set; }

        public decimal FactorForOral { get; set; }

        public decimal FactorForProject { get; set; }

        public decimal FactorForLab { get; set; }

    }
}
```

Lesson.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SchoolManagement.Data
{
    public class Lesson:BaseObject
```

```
    {
        public string Description { get; set; }
    }
}
```

LessonAssignement.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SchoolManagement.Data
{
    public class LessonAssignement
    {
        public Lesson Lesson { get; set; }

        public Department Department { get; set; }

        public override string ToString()
        {
            return Lesson.Name + " " + Department.Name;
        }
    }
}
```

Student.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SchoolManagement.Data
{
    public class Student:BaseObject
    {
        public string LastName { get; set; }
        public string FatherName { get; set; }
        public string RegistrationNumber { get; set; }
        public int YearOfRegistration { get; set; }
        public string PhoneNumber { get; set; }
        public string Email { get; set; }
        public List<LessonAssignement> RegisteredLessons { get; set; }
        public List<Absence> Absences { get; set; }

        public List<Grade> Grades { get; set; }
    }
}
```

Teacher.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace SchoolManagement.Data
{
    public class Teacher:BaseObject
    {
        public string LastName { get; set; }
        public string Office { get; set; }
        public int YearOfRegistration { get; set; }
        public string PhoneNumber { get; set; }
        public string Email { get; set; }
        public List<Lesson> Lessons { get; set; }
        public List<LessonAssignement> Assignements { get; set; }
    }
}
```

SchoolManagement (project)

DepartmentsControl.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using SchoolManagement.Data;

namespace SchoolManagement
{
    public partial class DepartmentsControl : UserControl
    {
        public DepartmentsControl()
        {
            InitializeComponent();

            LoadDepartments();

            LoadLessons();

        }

        private void LoadDepartments()
        {
            Database db = new Database();
            db.Load();
            listDepartments.DataSource = db.Departments.OrderBy(l=>l.Name) .ToList();
            listDepartments.DisplayMember = "Name";

            listLessons.SelectedIndexChanged += listLessons_SelectedIndexChanged;
        }



        private void btnAddLesson_Click(object sender, EventArgs e)
        {
```

```
            Database db = new Database();
            db.Load();
            int id = db.Departments.Count == 0 ? 1000 : db.Departments.Max(d => d.Id) + 1;
            db.Departments.Add(new Department { Name = textDepartment.Text, Id = id });
            db.Save();
            LoadDepartments();
        }

        private void btnRemove_Click(object sender, EventArgs e)
        {
            if (listDepartments.SelectedIndex < 0)
                return;
            var removeId = (listDepartments.SelectedItem as Department).Id;
            Database db = new Database();
            db.Load();
            db.Departments.Remove( db.Departments.Find(d=>d.Id == removeId ));
            db.Save();
            LoadDepartments();
        }
        void LoadLessons()
        {
            Database db = new Database();
            db.Load();
            listLessons.Items.AddRange(db.Lessons.OrderBy(l => l.Name).ToArray());
            listLessons.DisplayMember = "Name";
        }
        void listLessons_SelectedIndexChanged(object sender, EventArgs e)
        {
            Database db = new Database();
            db.Load();
            var lesson = listLessons.SelectedItem as Lesson;
            var definition =db.GradeFactors.FirstOrDefault(g => g.ForLesson.Id ==
             lesson.Id);
            if (definition == null)
            {
                textExam.Text = "";
                textVerbal.Text = "";
                textLab.Text = "";
                textProject.Text = "";
                return;
            }
            textExam.Text = definition.FactorForExams.ToString("0.00");
            textVerbal.Text = definition.FactorForOral.ToString("0.00");
            textLab.Text = definition.FactorForLab.ToString("0.00");
            textProject.Text = definition.FactorForProject.ToString("0.00");
        }

        private void btnAssign_Click(object sender, EventArgs e)
        {
            if (listLessons.SelectedIndex < 0)
                return;
            Database db = new Database();
            db.Load();
            var lesson = listLessons.SelectedItem as Lesson;
            var definition =db.GradeFactors.FirstOrDefault(g => g.ForLesson.Id ==
lesson.Id);
            if (definition == null)
            {
                db.GradeFactors.Add(new GradeDefinition {
                    ForLesson = lesson,
                    FactorForExams =decimal.Parse( textExam.Text ),
                    FactorForLab= decimal.Parse(textLab.Text),
```

```
                    FactorForOral= decimal.Parse(textVerbal.Text),
                    FactorForProject = decimal.Parse(textProject.Text)
                });
            }
            else
            {
                    definition.FactorForExams =decimal.Parse( textExam.Text );
                    definition.FactorForLab = decimal.Parse(textLab.Text);
                    definition.FactorForOral = decimal.Parse(textVerbal.Text);
                    definition.FactorForProject = decimal.Parse(textProject.Text);
            }
            db.Save();
        }
    }
}
```

Extensions.cs

```
using SchoolManagement.Data;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SchoolManagement
{
    public class AbsenceGridItem
    {
        public DateTime StartDate { get; set; }

        public DateTime EndDate { get; set; }

        public string LessonName { get; set; }

        public string DeparmentName { get; set; }

        public int Total { get; set; }
    }
    public static class Extensions
    {
        public static List<AbsenceGridItem> ToFlat(this List<Absence> d)
        {
            List<AbsenceGridItem> data = new List<AbsenceGridItem>();
            foreach(var item in d)
                data.Add(  new AbsenceGridItem(){ DeparmentName =
item.RegisteredLesson.Department.Name,
                                    LessonName = item.RegisteredLesson.Lesson.Name,
                                    EndDate = item.EndDate,
                                    StartDate =item.StartDate,
                                    Total = item.EndDate.Subtract(item.StartDate).Days +1
                });
            return data;
        }
        public static List<dynamic> ToFlat(this List<Grade> d)
        {
            List<dynamic> data = new List<dynamic>();
            foreach (var item in d)
            {
                data.Add(new
                {
```

```
                    Lesson = item.ForLesson.Lesson.Name,
                    FreeField = item.ValueForOral,
                    Exam = item.Value,
                    Lab = item.ValueForLab,
                    Project = item.ValueForProject,
                    Passed = CalculateResult(item)
                });
            }
            return data;
        }

        private static string CalculateResult(Grade grade)
        {
            Database db = new Database();
            db.Load();
            var lesson = grade.ForLesson;
            var definition = db.GradeFactors.FirstOrDefault(g => g.ForLesson.Id ==
lesson.Lesson.Id);
            if (definition == null)
                return " ";
            decimal result = 0.0M;

            if (definition.FactorForExams > 0)
            {
                result += ( definition.FactorForExams/100) * grade.Value;
            }
            if (definition.FactorForLab > 0)
            {
                result += (definition.FactorForLab/ 100) * grade.ValueForLab;
            }
            if (definition.FactorForOral > 0)
            {
                result += (definition.FactorForOral/ 100) * grade.ValueForOral;
            }
            if (definition.FactorForProject > 0)
            {
                result += (definition.FactorForProject/ 100) * grade.ValueForProject;
            }
            return result>5? "ΕΠΙΤΥΧΩΝ ":"ΠΑΡΑΠΕΜΠΕΤΑΙ ";
        }
    }
}
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using SchoolManagement.Data;

namespace SchoolManagement
{
    public partial class Form1 : Form
    {
```

```
        public Form1()
        {
            InitializeComponent();

            LessonsLoader.LoadLessonsFromText();
        }

        private void button1_Click(object sender, EventArgs e)
        {

        }

        private void LoadNames()
        {
            Database db = new Database();
            db.Load();

            AutoCompleteStringCollection strings = new AutoCompleteStringCollection();
            strings.AddRange(db.Students.Select(n => n.Name).ToArray());

        }
    }
}
```

LessonsLoader.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using SchoolManagement.Data;
namespace SchoolManagement
{
    class LessonsLoader
    {
        public static bool LoadLessonsFromText()
        {
            Database db = new Database();
            db.Load();

            if (db.Lessons.Count > 0)
                return false;
            var lines = File.ReadLines("lessons.txt");
            int idseed = 1000;
            foreach (string line in lines)
            {
                db.Lessons.Add(new Lesson { Id = idseed, Name = line,Description = line });
                idseed++;
            }
            db.Save();
            return true;
        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SchoolManagement
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

ReportControl.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using SchoolManagement.Data;
using System.IO;

namespace SchoolManagement
{
    public partial class ReportControl : UserControl
    {
        public ReportControl()
        {
            InitializeComponent();
            LoadLessons();
        }

        private void LoadLessons()
        {
            Database db = new Database();
            db.Load();

            var lessons = db.Students.SelectMany(s => s.RegisteredLessons).
GroupBy(f=>f.Lesson.Name).Select(k=>k.Key).ToList();
            cboLessons.DisplayMember = "Name";
            cboLessons.DataSource = lessons;
        }

        private void cboLessons_SelectedIndexChanged(object sender, EventArgs e)
        {
            Database db = new Database();
```

```
            db.Load();

            var lesson = cboLessons.SelectedItem.ToString();

            var departments = db.Students.SelectMany(s => s.RegisteredLessons).
                                  Where(s=>s.Lesson.Name == lesson).
                                  Select(k => k.Department).
                                  GroupBy(department => department.Id).
                                  Select(group => group.First()).
                                  ToList();
            cboDepartment.DisplayMember = "Name";
            cboDepartment.DataSource = departments;
        }

        private void btnGrades_Click(object sender, EventArgs e)
        {
            Database db = new Database();
            db.Load();
            var lesson = cboLessons.SelectedItem.ToString();

            var department = cboDepartment.SelectedItem as Department;


            var gradesQuery =   from student in db.Students
                                where student.Grades.Any(g => g.ForLesson.Lesson.Name ==
lesson && g.ForLesson.Department.Name == department.Name)
                                let grade =
student.Grades.First(g=>g.ForLesson.Lesson.Name== lesson && g.ForLesson.Department.Name ==
department.Name)
                                select new { Name =" "+ student.Name+" ",
                                             LastName =student.LastName+" ",
                                             Lesson =grade.ForLesson.Lesson.Name+" ",
                                             Department =grade.ForLesson.Department.Name+"
",
                                             Value=grade.Value.ToString()+" " ,
                                             ValueForLab=grade.ValueForLab.ToString()+" " ,

ValueForProject=grade.ValueForProject.ToString()+" ",
                                             ValueForOral=grade.ValueForOral.ToString()+"
"};


            // the data from the query
           dataGridView1.DataSource = gradesQuery.ToList();


            // change header names
            dataGridView1.Columns[0].HeaderText = "Ονομα ";
            dataGridView1.Columns[1].HeaderText = "Επίθετο ";
            dataGridView1.Columns[2].HeaderText = "Μάθημα ";
            dataGridView1.Columns[3].HeaderText = "Τμήμα ";
            dataGridView1.Columns[4].HeaderText = "Exams ";
            dataGridView1.Columns[5].HeaderText = "Lab ";
            dataGridView1.Columns[6].HeaderText = "Project ";
            dataGridView1.Columns[7].HeaderText = "Free-field ";

        }
        private void btnAbsences_Click(object sender, EventArgs e)
        {
            Database db = new Database();
            db.Load();
            var lesson = cboLessons.SelectedItem.ToString();
```

```
            var department = cboDepartment.SelectedItem as Department;

            var absQuery = from student in db.Students
                           where student.Absences.Any(g => g.RegisteredLesson.Lesson.Name
== lesson && g.RegisteredLesson.Department.Name == department.Name)

                           let abs = student.Absences.Last(g =>
g.RegisteredLesson.Lesson.Name == lesson)
                               select new
                               {
                                   Name = " " + student.Name + " ",
                                   LastName = student.LastName + " ",
                                   Lesson = abs.RegisteredLesson.Lesson.Name + " ",
                                   Department = abs.RegisteredLesson.Department.Name + " ",
                                   StartDate = abs.StartDate.ToString() + " ",
                                   EndDate = abs.EndDate.ToString() + " ",
                                   Total = abs.EndDate.Subtract(abs.StartDate).Days + 1
                               };


            foreach(var abs in absQuery)
            {
                dataGridView1.DataSource = absQuery.ToList();
            };


            dataGridView1.Columns[0].HeaderText = "Ονομα ";
            dataGridView1.Columns[1].HeaderText = "Επίθετο ";
            dataGridView1.Columns[2].HeaderText = "Μάθημα ";
            dataGridView1.Columns[3].HeaderText = "Τμήμα ";
            dataGridView1.Columns[4].HeaderText = "StartDate ";
            dataGridView1.Columns[5].HeaderText = "EndDate ";
            dataGridView1.Columns[6].HeaderText = "Total  ";
        }
        private void btnResults_Click(object sender, EventArgs e)
        {
            Database db = new Database();
            db.Load();
            var lesson = cboLessons.SelectedItem.ToString();

            var department = cboDepartment.SelectedItem as Department;



            var resultQuery = from student in db.Students
                              where student.Grades.Any(g => g.ForLesson.Lesson.Name ==
lesson && g.ForLesson.Department.Name == department.Name)
                              let grade = student.Grades.First(g => g.ForLesson.Lesson.Name
== lesson)
                              select new
                              {
                                  Name = " "+ student.Name+" ",
                                  LastName = student.LastName+" ",
                                  Lesson = grade.ForLesson.Lesson.Name + " ",
                                  Department = grade.ForLesson.Department.Name + " ",
                                  Passed = CalculateResult(grade)
                              };

            dataGridView1.DataSource = resultQuery.ToList();
```

```
            dataGridView1.Columns[0].HeaderText = "Ονομα ";
            dataGridView1.Columns[1].HeaderText = "Επίθετο ";
            dataGridView1.Columns[2].HeaderText = "Μάθημα ";
            dataGridView1.Columns[3].HeaderText = "Τμήμα ";
            dataGridView1.Columns[4].HeaderText = "Αποτέλεσμα ";
        }
        private static string CalculateResult(Grade grade)
        {
            Database db = new Database();
            db.Load();
            var lesson = grade.ForLesson;
            var definition = db.GradeFactors.FirstOrDefault(g => g.ForLesson.Id ==
lesson.Lesson.Id);
            if (definition == null)
                return " ";
            decimal result = 0.0M;

            if (definition.FactorForExams > 0)
            {
                result += (definition.FactorForExams / 100) * grade.Value;
            }
            if (definition.FactorForLab > 0)
            {
                result += (definition.FactorForLab / 100) * grade.ValueForLab;
            }
            if (definition.FactorForOral > 0)
            {
                result += (definition.FactorForOral / 100) * grade.ValueForOral;
            }
            if (definition.FactorForProject > 0)
            {
                result += (definition.FactorForProject / 100) * grade.ValueForProject;
            }
            return result > 5 ? "ΕΠΙΤΥΧΩΝ " : "ΠΑΡΑΠΕΜΠΕΤΑΙ ";
        }
        private void ToCsV(DataGridView dGV, string filename)
        {
            string stOutput = " ";
            // Export titles:
            string sHeaders = " ";

            for (int j = 0; j < dGV.Columns.Count; j++)
                sHeaders = sHeaders.ToString() +
Convert.ToString(dGV.Columns[j].HeaderText) + "\t";
                stOutput += sHeaders + "\r\n";
            // Export data.
            for (int i = 0; i < dGV.RowCount; i++)
            {
                string stLine = " ";
                for (int j = 0; j < dGV.Rows[i].Cells.Count; j++)
                    stLine = stLine.ToString() +
Convert.ToString(dGV.Rows[i].Cells[j].Value) + "\t";
                    stOutput += stLine + "\r\n";
            }
            Encoding utf16 = Encoding.Default;// GetEncoding
            byte[] output = utf16.GetBytes(stOutput);
            FileStream fs = new FileStream(filename, FileMode.Create);
            BinaryWriter bw = new BinaryWriter(fs);
            bw.Write(output, 0, output.Length); //write the encoded file
            bw.Flush();
            bw.Close();
            fs.Close();
```

```
        }

        private void button1_Click(object sender, EventArgs e)
        {
            SaveFileDialog sfd = new SaveFileDialog();
            sfd.Filter = "Excel Documents (*.xls)|*.xls";
            sfd.FileName = "export.xls";
            if (sfd.ShowDialog() == DialogResult.OK)
            {
                ToCsV(dataGridView1, sfd.FileName);
            }
        }
    }
}
```

StudentControl.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using SchoolManagement.Data;

namespace SchoolManagement
{
    public partial class StudentControl : UserControl
    {
        private bool CreateNew = false;
        public StudentControl()
        {
            InitializeComponent();
            dataGridView1.SelectionChanged += dataGridView1_SelectionChanged;
            dataGridView1.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
            dataGridView1.ReadOnly = true;

            dataGridAbsences.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
            dataGridAbsences.ReadOnly = true;

            cboLessonForGrades.SelectedIndexChanged +=
cboLessonForGrades_SelectedIndexChanged;
            LoadLessons();
        }

        void dataGridView1_SelectionChanged(object sender, EventArgs e)
        {
            if (dataGridView1.SelectedRows.Count <1)
                return;
            var item =(Student) dataGridView1. SelectedRows[0].DataBoundItem;
            textName.Text = item.Name;
            textLastName.Text = item.LastName;
            textEmail.Text = item.Email;
            textFatherName.Text = item.FatherName;
            textRegistration.Text = item.RegistrationNumber;
            textYear.Text = item.YearOfRegistration.ToString();
            textPhone.Text = item.PhoneNumber;
```

```
            LoadLessonsOfStudent(item);

            LoadAbsencesOfStudent(item);

            LoadGrades();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            if (dataGridView1.SelectedRows.Count < 1 || CreateNew)
                AddStudent();
            else
            {
                UpdateStudent();
            }
        }

        private void AddStudent()
        {
            Database db = new Database();
            db.Load();
            int maxid = 0;
            if (db.Students.Count == 0)
                maxid = 1000; // initial id value...
            else
            {
                // find the max value.
                var ids = from s in db.Students
                          orderby s.Id descending
                          select s.Id;

                maxid = ids.First() + 1;
            }
            Student student = new Student();
            student.Id = maxid;
            student.Name = textName.Text;
            student.LastName = textLastName.Text;
            student.FatherName = textFatherName.Text;
            student.Email = textEmail.Text;
            student.PhoneNumber = textPhone.Text;
            student.RegistrationNumber = textRegistration.Text;
            student.YearOfRegistration = int.Parse(textYear.Text);

            db.Students.Add(student);
            db.Save();

            CreateNew = false;
            // refresh the grid
            Search();
        }
        private void UpdateStudent()
        {
            var item = (Student)dataGridView1.SelectedRows[0].DataBoundItem;
            Database db = new Database();
            db.Load();
            var existing = db.Students.First(s => s.Id == item.Id);

            existing.Name = textName.Text;
            existing.LastName = textLastName.Text;
            existing.FatherName = textFatherName.Text;
            existing.Email = textEmail.Text;
            existing.PhoneNumber = textPhone.Text;
```

```
            existing.RegistrationNumber = textRegistration.Text;
            existing.YearOfRegistration = int.Parse(textYear.Text);
            db.Save();

            Search();
        }
        private void buttonSearch_Click(object sender, EventArgs e)
        {
            Search();
        }

        private void Search()
        {
            Database db = new Database();
            db.Load();

            var data = db.Students.Where(s =>
s.LastName.StartsWith(textSearch.Text)).ToList();
            dataGridView1.DataSource = null;
            dataGridView1.DataSource = data;
        }

        private void buttonNew_Click(object sender, EventArgs e)
        {
            textName.Text = "";
            textLastName.Text = "";
            textEmail.Text = "";
            textFatherName.Text = "";
            textRegistration.Text = "";
            textYear.Text = "";
            textPhone.Text = "";
            CreateNew = true;
        }

        void LoadLessons()
        {
            Database db = new Database();
            db.Load();
            cboLessons.Items.AddRange(db.Lessons.OrderBy(l=>l.Name). ToArray());
            cboLessons.DisplayMember = "Name";
        }
        private void LoadLessonsOfStudent(Student s)
        {
            Database db = new Database();
            db.Load();
            var dbstudent = db.Students.First(dbs => dbs.Id == s.Id);
            if(dbstudent.RegisteredLessons.Count>0)
            {
                listLessons.DataSource =
dbstudent.RegisteredLessons.Select(l=>l.Lesson).OrderBy(l=>l.Name). ToList();
                listLessons.DisplayMember = "Name";
            }
            else
            {
                listLessons.DataSource = null;
            }
        }
        private void btnAddLesson_Click(object sender, EventArgs e)
        {

            var item = (Student)dataGridView1.SelectedRows[0].DataBoundItem;
            Database db = new Database();
```

```
            db.Load();

            var lesson = cboLessons.SelectedItem as Lesson;

            var student = db.Students.First(s => s.Id == item.Id);

            if (student.RegisteredLessons.Any(lessondb => lessondb.Lesson!=null &&
lessondb.Lesson.Name == lesson.Name))
            {

                return;
            }

            student.RegisteredLessons.Add(new LessonAssignement { Lesson = lesson,
Department = new Department() });

            db.Save();

            LoadLessonsOfStudent(student);
        }

        private void listLessons_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (listLessons.SelectedItem == null)
                return;
            Database db = new Database();
            db.Load();
            var gridStudent = (Student)dataGridView1.SelectedRows[0].DataBoundItem;
            // load the student from database
            var student = db.Students.First(s => s.Id == gridStudent.Id);

            listDepartments.Items.Clear();
            int lessonId = (listLessons.SelectedItem as Lesson).Id;
            //select the departments that are assigned to teachers for a given lesson id...
            var departments = from teacher in db.Teachers
                              from assignement in teacher.Assignements
                              where assignement.Lesson.Id == lessonId &&
db.Lessons.Any(g=>g.Id == lessonId)
                              select assignement.Department;

            if (departments.Count() > 0)
            {
                var data = departments.OrderBy(d => d.Name).ToArray();
                listDepartments.Items.AddRange(data);
                listDepartments.DisplayMember = "Name";
            }
            // get the lesson from the list
            var lesson = listLessons.SelectedItem as Lesson;

            // iterate to all the departments
            for (int i = 0; i < listDepartments.Items.Count; i++)
            {
                // get the current department in the iteration
                var department = listDepartments.Items[i] as Department;

                //  check if the student is registered to a department for this lesson
                if (student.RegisteredLessons.Any(a => a.Department.Id == department.Id &&
a.Lesson.Id == lesson.Id))
                {

                    listDepartments.SetItemChecked(i, true);
                }
```

```
                else
                    listDepartments.SetItemChecked(i, false);

            }
        }

        private void btnRemove_Click(object sender, EventArgs e)
        {
            // check if something is selected
            if (listLessons.SelectedItem == null)
                return;

            var item = (Student)dataGridView1.SelectedRows[0].DataBoundItem;
            Database db = new Database();
            db.Load();

            var lesson = listLessons.SelectedItem as Lesson;
            //  load the student
            var student = db.Students.First(s => s.Id == item.Id);


            student.RegisteredLessons.Remove(student.RegisteredLessons.First(l => l.Lesson.
Id == lesson.Id));

            db.Save();
            // load the list with the lessons of the teacher
            LoadLessonsOfStudent(student);
        }

        private void btnAssign_Click(object sender, EventArgs e)
        {
            Database db = new Database();
            db.Load();
            var gridStudent = (Student)dataGridView1.SelectedRows[0].DataBoundItem;

            var student = db.Students.First(s => s.Id == gridStudent.Id);

            var lesson = listLessons.SelectedItem as Lesson;
            //find the registration of the student for this lesson
            var registration = student.RegisteredLessons.Find(l => l.Lesson.Id ==
lesson.Id);
            // iterate to all the departments and find the ones that are selected
            for (int i = 0; i < listDepartments.Items.Count; i++)
            {
                //get the department from the check box list
                var department = listDepartments.Items[i] as Department;

                if (listDepartments.GetItemChecked(i))
                {
                    //update the registration with the selected department
                    registration.Department = department;
                    break;
                }
            }

            db.Save();
        }

        private void LoadAbsencesOfStudent(Student s)
        {
            Database db = new Database();
            db.Load();
```

```
            var dbstudent = db.Students.First(dbs => dbs.Id == s.Id);
            if (dbstudent.RegisteredLessons.Count > 0)
            {
                cboLessonsForAbsence.DataSource = dbstudent.RegisteredLessons.Select(l =>
l.Lesson).OrderBy(l => l.Name).ToList();
                cboLessonsForAbsence.DisplayMember = "Name";
            }
            else
            {
                cboLessonsForAbsence.DataSource = null;
            }
            dataGridAbsences.DataSource = dbstudent.Absences.ToFlat();
        }

        private void btnAddAbsence_Click(object sender, EventArgs e)
        {
            Database db = new Database();
            db.Load();

            var gridStudent = (Student)dataGridView1.SelectedRows[0].DataBoundItem;
            //load student from db
            var dbstudent = db.Students.First(dbs => dbs.Id == gridStudent.Id);
            //find the lesson for which the absence will be added
            var lesson = dbstudent.RegisteredLessons.Where(l => l.Lesson.Id ==

(cboLessonsForAbsence.SelectedItem as Lesson).Id).First();

            dbstudent.Absences.Add( new Absence{StartDate =dateTimePicker1.Value,
                                                EndDate = dateTimePicker2.Value,
                                                RegisteredLesson =lesson});

            db.Save();

            LoadAbsencesOfStudent(dbstudent);
        }

        private void btnRemoveAbsence_Click(object sender, EventArgs e)
        {
            Database db = new Database();
            db.Load();

            var gridStudent = (Student)dataGridView1.SelectedRows[0].DataBoundItem;

            var dbstudent = db.Students.First(dbs => dbs.Id == gridStudent.Id);

            var absence = (AbsenceGridItem)dataGridAbsences.SelectedRows[0].DataBoundItem;

            dbstudent.Absences.Remove( dbstudent.Absences.First( ab=>ab.StartDate ==
absence.StartDate &&
                                                                 ab.EndDate ==
absence.EndDate &&

ab.RegisteredLesson.Lesson.Name == absence.LessonName) );

            db.Save();

            //refresh the grid
            LoadAbsencesOfStudent(dbstudent);
        }
        void LoadGrades()
```

```
        {
            Database db = new Database();
            db.Load();
            var gridStudent = (Student)dataGridView1.SelectedRows[0].DataBoundItem;

            var dbstudent = db.Students.First(dbs => dbs.Id == gridStudent.Id);
            if (dbstudent.RegisteredLessons.Count > 0)
            {
                cboLessonForGrades.DataSource = dbstudent.RegisteredLessons;
                cboLessonForGrades.SelectedIndex = 0;
            }


            if (dbstudent.Grades.Count == 0)
            {
                foreach(var lesson in dbstudent.RegisteredLessons){
                    dbstudent.Grades.Add(new Grade() { ForLesson = lesson });
                }
                db.Save();
            }
            else if(dbstudent.RegisteredLessons.Count > dbstudent.Grades.Count)
            {
                foreach (var lesson in dbstudent.RegisteredLessons)
                {
                    if(!dbstudent.Grades.Any( g=>g.ForLesson.Lesson.Id ==
lesson.Lesson.Id))
                    dbstudent.Grades.Add(new Grade() { ForLesson = lesson });
                }
                db.Save();
            }


            if (dbstudent.Grades.Count > 0)
            {
                dataGridGrades.DataSource = dbstudent.Grades.ToFlat();

            }
            else
            {
                dataGridGrades.DataSource = null;
            }
        }

        void cboLessonForGrades_SelectedIndexChanged(object sender, EventArgs e)
        {
            Database db = new Database();
            db.Load();
            var gridStudent = (Student)dataGridView1.SelectedRows[0].DataBoundItem;

            var dbstudent = db.Students.First(dbs => dbs.Id == gridStudent.Id);
            var lesson = cboLessonForGrades.SelectedItem as LessonAssignement;
            var grade = dbstudent.Grades.Where(g => g.ForLesson.Lesson.Id ==
lesson.Lesson.Id).FirstOrDefault();
            if (grade == null)
                return;


            textExam.Text = grade.Value.ToString();
            textLab.Text = grade.ValueForLab.ToString();
            textVerbal.Text = grade.ValueForOral.ToString();
            textProject.Text = grade.ValueForProject.ToString();
```

```
            EnableGradeFields();

        }

        private void btnSetGrade_Click(object sender, EventArgs e)
        {
            Database db = new Database();
            db.Load();
            var gridStudent = (Student)dataGridView1.SelectedRows[0].DataBoundItem;

            var dbstudent = db.Students.First(dbs => dbs.Id == gridStudent.Id);
            var lesson = cboLessonForGrades.SelectedItem as LessonAssignement;
            var grade = dbstudent.Grades.Where(g => g.ForLesson.Lesson.Id ==
lesson.Lesson.Id).First();

                    grade.Value = decimal.Parse(textExam.Text);
                    grade.ValueForLab = decimal.Parse(textLab.Text);
                    grade.ValueForOral = decimal.Parse(textVerbal.Text);
                    grade.ValueForProject = decimal.Parse(textProject.Text);

            if (dbstudent.Grades.Count > 0)
            {
                dataGridGrades.DataSource = dbstudent.Grades.ToFlat();
            }
            db.Save();
        }

        void EnableGradeFields()
        {
            Database db = new Database();
            db.Load();
            var lesson = cboLessonForGrades.SelectedItem as LessonAssignement;
            var definition = db.GradeFactors.FirstOrDefault(g => g.ForLesson.Id ==
lesson.Lesson.Id);

            textProject.Enabled = textVerbal.Enabled = textProject.Enabled =
textLab.Enabled = true;
            btnSetGrade.Enabled = true;
            if (definition == null)
            {
                MessageBox.Show("Δεν εχουν οριστει συντελεστες βαρυτητας για το μαθημα.",
"Προσοχή", MessageBoxButtons.OK, MessageBoxIcon.Stop);
                textProject.Enabled = textVerbal.Enabled = textProject.Enabled =
textLab.Enabled = false;
                btnSetGrade.Enabled = false;
                return;
            }
            if (definition.FactorForExams==0)
            {
                textExam.Enabled = false;
            }
            if (definition.FactorForLab== 0)
            {
                textLab.Enabled = false;
            }
            if (definition.FactorForOral== 0)
            {
                textVerbal.Enabled = false;
            }
            if (definition.FactorForProject== 0)
            {
                textProject.Enabled = false;
```

```
            }
        }
    }
}
```

TeachersControl.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using SchoolManagement.Data;

namespace SchoolManagement
{
    public partial class TeachersControl : UserControl
    {
        private bool CreateNew = false;
        public TeachersControl()
        {
            InitializeComponent();
            dataGridView1.SelectionChanged += dataGridView1_SelectionChanged;
            dataGridView1.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
            dataGridView1.ReadOnly = true;

            listLessons.SelectedIndexChanged += listLessons_SelectedIndexChanged;

            LoadLessons();


            LoadDepartments();
        }


        void LoadLessons()
        {
            Database db = new Database();
            db.Load();
            cboLessons.Items.AddRange(db.Lessons.ToArray());
            cboLessons.DisplayMember = "Name";
        }
        void LoadDepartments()
        {
            Database db = new Database();
            db.Load();
            listDepartments . Items.AddRange(db.Departments.ToArray());
            listDepartments.DisplayMember = "Name";
        }
        void dataGridView1_SelectionChanged(object sender, EventArgs e)
        {
            if (dataGridView1.SelectedRows.Count < 1)
                return;
            var item = (Teacher)dataGridView1.SelectedRows[0].DataBoundItem;
            textName.Text = item.Name;
            textLastName.Text = item.LastName;
```

```
            textEmail.Text = item.Email;
            textOffice.Text = item.Office;
            textPhone.Text = item.PhoneNumber;

            LoadLessonsOfTeacher(item);
        }
        private void button1_Click(object sender, EventArgs e)
        {
            if (dataGridView1.SelectedRows.Count < 1 || CreateNew)
                AddTeacher();
            else
            {
                UpdateTeacher();
            }
        }

        private void AddTeacher()
        {
            Database db = new Database();
            db.Load();
            int maxid = 0;
            if (db.Teachers.Count == 0)
                maxid = 1000; // initial id value...
            else
            {
                // try to find the max value.
                var ids = from s in db.Teachers
                          orderby s.Id descending
                          select s.Id;

                maxid = ids.First() + 1;
            }
            Teacher t = new Teacher();
            t.Id = maxid;
            t.Name = textName.Text;
            t.LastName = textLastName.Text;
            t.Office = textOffice.Text;
            t.Email = textEmail.Text;
            t.PhoneNumber = textPhone.Text;

            db.Teachers.Add(t);
            db.Save();

            CreateNew = false;
            // refresh the grid
            Search();
        }
        private void UpdateTeacher()
        {
            var item = (Teacher)dataGridView1.SelectedRows[0].DataBoundItem;
            Database db = new Database();
            db.Load();
            var existing = db.Teachers.First(s => s.Id == item.Id);

            existing.Name = textName.Text;
            existing.LastName = textLastName.Text;
            existing.Office = textOffice.Text;
            existing.Email = textEmail.Text;
            existing.PhoneNumber = textPhone.Text;
            db.Save();

            // refresh the grid
```

```
            Search();
        }
        private void buttonSearch_Click(object sender, EventArgs e)
        {
            Search();
        }

        private void Search()
        {
            Database db = new Database();
            db.Load();

            var data = db.Teachers.Where(s =>
s.LastName.StartsWith(textSearch.Text)).ToList();
            dataGridView1.DataSource = null;
            dataGridView1.DataSource = data;
        }

        private void buttonNew_Click(object sender, EventArgs e)
        {
            textName.Text = "";
            textLastName.Text = "";
            textEmail.Text = "";
            textOffice.Text = "";
            textPhone.Text = "";
            CreateNew = true;
        }

        private void button1_Click_1(object sender, EventArgs e)
        {

        }

        private void btnAddLesson_Click(object sender, EventArgs e)
        {
            // the selected teacher from the grid
            var item = (Teacher)dataGridView1.SelectedRows[0].DataBoundItem;
            Database db = new Database();
            db.Load();
            // the selected lesson from the combo box
            var lesson = cboLessons.SelectedItem as Lesson;

            var teacher = db.Teachers.First(s => s.Id == item.Id);
            // add the lesson if it does not exist
            if (teacher.Lessons.Any(lessondb => lessondb.Id == lesson.Id))
            {

                return;
            }

            teacher.Lessons.Add(lesson);
            // update the database
            db.Save();
            // load the list with the lessons of the teacher
            LoadLessonsOfTeacher(teacher);
        }

        private void LoadLessonsOfTeacher(Teacher teacher)
        {
            listLessons.DataSource = teacher.Lessons;
            listLessons.DisplayMember = "Name";
        }
```

```
        private void btnRemove_Click(object sender, EventArgs e)
        {
            if (listLessons.SelectedItem == null)
                return;

            var item = (Teacher)dataGridView1.SelectedRows[0].DataBoundItem;
            Database db = new Database();
            db.Load();

            var lesson = listLessons.SelectedItem as Lesson;

            var teacher = db.Teachers.First(s => s.Id == item.Id);

            teacher.Lessons.Remove( teacher.Lessons.First( l=>l.Id == lesson.Id));

            db.Save();
            // load the list with the lessons of the teacher
            LoadLessonsOfTeacher(teacher);
        }
        void listLessons_SelectedIndexChanged(object sender, EventArgs e)
        {
            Database db = new Database();
            db.Load();
            var gridTeacher = (Teacher)dataGridView1.SelectedRows[0].DataBoundItem;
            // load the teacher from database
            var teacher = db.Teachers.First(s => s.Id == gridTeacher.Id);

            listDepartments.ClearSelected();

            // iterate to all the departments
            for (int i = 0; i < listDepartments.Items.Count; i++)
            {
                // get the lesson from the list
                var lesson = listLessons.SelectedItem as Lesson;
                // get the current department in the iteration
                var department = listDepartments.Items[i] as Department;

                //  check if the lesson is  assigned to a department for this teacher
                if (teacher.Assignements.Any(a => a.Department.Id == department.Id &&
a.Lesson.Id == lesson.Id))
                {
                    listDepartments.SetItemChecked(i, true);
                }
                else
                    listDepartments.SetItemChecked(i, false);

            }
        }
        private void btnAssign_Click(object sender, EventArgs e)
        {
            Database db = new Database();
            db.Load();
            var gridTeacher = (Teacher)dataGridView1.SelectedRows[0].DataBoundItem;
            // load the teacher from database
            var teacher = db.Teachers.First(s => s.Id == gridTeacher.Id);

            var lesson = listLessons.SelectedItem as Lesson;

            // iterate to all the departments and find the ones that are selected
            for (int i = 0; i < listDepartments.Items.Count; i++)
            {
```

```
                //get the department from the check box list
                var department = listDepartments.Items[i] as Department;

                if (listDepartments.GetItemChecked(i))
                {

                    if (!teacher.Assignements.Any(a => a.Department.Id == department.Id &&
a.Lesson.Id == lesson.Id))
                    {

                        teacher.Assignements.Add(new LessonAssignement { Department =
department, Lesson = lesson });
                    }
                }
                else
                    teacher.Assignements.RemoveAll(p=>p.Lesson.Id == lesson.Id &&
department.Id == p.Department.Id);
            }
            db.Save();
        }
    }
}
```

Βιβλιογραφία - Ιστοσελίδες

- Developer// Step by step - Microsoft Visual C# 2013
  Intermediate – John Sharp

- Windows Forms in Action – Erik Brown
  Second Edition of Windows Forms Programming with C#

- .NET Book Zero
  What the C or C++ Programmer Needs to Know about C# and the .NET Framework
  Charles Petzold, 2006-2007 - http://www.charlespetzold.com

- FUNDAMENTALS OF COMPUTER PROGRAMMING WITH C#
  (The Bulgarian C# Programming Book)
  Svetlin Nakov & Co., 2013

- https://en.wikibooks.org/wiki/C_Sharp_Programming

- https://msdn.microsoft.com/en-us/default.aspx

- http://stackoverflow.com/

- https://en.wikipedia.org/wiki/Comma-separated_values

- http://www.c-sharpcorner.com/