Microsoft Hellas

ΑΠΟ ΤΗ JAVA ΣΤΗ C#

Καθηγητού: Ι.-Χ. ΠΑΝΑΓΙΩΤΟΠΟΥΛΟΥ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ

Το παρόν σύγγραμμα είναι ένα αυτοτελές εκπαιδευτικό βοήθημα για εκείνους που γνωρίζουν JAVA (2++) και επιθυμούν να μάθουν ταχύρυθμα αλλά ικανοποιητικά τη γλώσσα C# της Microsoft.

Τα δώδεκα κεφάλαια του συγγράμματος συγκροτούνται από διαφάνειες και αντίστοιχη ύλη καθώς επίσης και από ορισμένες ασκήσεις. Όλα τα προγράμματα σε JAVA και C# έχουν δοκιμαστεί και υπάρχει αντιστοίχηση μεταξύ τους με τον πιο απλό τρόπο για καθαρά εκπαιδευτικούς λόγους.

Το παρόν σύγγραμμα είναι προϊόν επιστημονικού έργου μεταξύ της Microsoft Hellas και του Κέντρου Ερευνών του Πανεπιστημίου Πειραιώς με επιστημονικό υπεύθυνο τον συγγραφέα καθηγητή Ι-Χ. Παναγιωτόπουλο και μετέχοντα τον επιστημονικό συνεργάτη κ. Παύλο Πετραντωνάκη. Ο δεύτερος εκτός από την επιμέλεια των κειμένων έχει κάνει και εργασία ουσίας και συμμετείχε ενεργά σε όλα τα στάδια του έργου.

Ελπίζουμε το σύγγραμμα αυτό να βοηθήσει αποτελεσματικά τους φοιτητές ΑΕΙ και ΤΕΙ, τμημάτων σχετικών με την κοινωνία της πληροφορίας για το αποφασιστικό πέρασμα από το κλασσικό, την JAVA, στο νέο και καινοτόμο, την C#, ένα από τα πλέον πρόσφατα δημιουργήματα του Διαδικτυακού προγραμματισμού της Microsoft,με χρήση σε desktop εφαρμογές, web εφαρμογές και εφαρμογές σε smart phones και PDA' s.

Είναι τιμή μας που η Microsoft μας προτίμησε γι' αυτό το έργο. Ελπίζουμε το βιβλίο αυτό να έχει καλή τύχη στην τριτοβάθμια εκπαίδευση και να φανεί χρήσιμο σε διδάσκοντες και διδασκόμενους

Πειραιάς, Ιούνιος 2003

Ο ΣΥΓΓΡΑΦΕΑΣ
Ι-Χ. Παναγιωτόπουλος
Καθηγητής

ΚΕΦΑΛΑΙΟ 1ο

1. ΕΙΣΑΓΩΓΗ ΣΤΟ ΔΙΑΔΙΚΤΥΑΚΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

1.1 ΠΑΡΑΔΟΣΙΑΚΕΣ ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Στη δεκαετία του 50 φάνηκαν οι πρώτες γλώσσες προγραμματισμού, οι οποίες βασίζονταν σε ένα πρόγραμμα μεταγλώττισης (compiler) ο οποίος παρήγαγε obj προγράμματα (object). Εκείνη την εποχή τα mainframes που υπήρχαν εκτελούσαν άμεσα τα obj προγράμματα χωρίς βέβαια την ανάγκη ύπαρξης αντιστοιχών εκτελέσιμων προγραμμάτων (exe,com κτλ) όπως συμβαίνει στην μικροπληροφορική σήμερα.

Αρχικά εμφανίστηκαν 3 γλώσσες προγραμματισμού, η COBOL, η FORTRAN και η RPG. Η COBOL χρησιμοποιήθηκε αποκλειστικά για οικονομικές-λογιστικές εφαρμογές, όπου υπερτερούσε των άλλων σε αυτοματοποιημένες διαδικασίες που διέθετε για προσπέλαση των πρωτόγονων τότε βάσεων δεδομένων. Η FORTRAN κυριάρχησε σε επιστημονικές-τεχνικές εφαρμογές και έθεσε την αρχή μεταβλητών και εντολών που θα είχαν οι μετέπειτα γλώσσες προγραμματισμού. Η RPG ήταν μια μικρή, ευέλικτη γλώσσα που έβγαλε η IBM, αποκλειστικά για προγράμματα με λίγη απλή λογική και μεγάλο όγκο εξόδου εκτυπώσεων.

Τα βασικά ελαττώματα των πρώτων αυτών γλωσσών προγραμματισμού φάνηκαν από την δεκαετία του 70 και μετά με την κρίση του software (software crisis) του δομημένου προγραμματισμού. Τότε η COBOL μεταλλάχτηκε στη COBOL 74, η FORTRAN περνώντας από διάφορα στάδια, εκδόσεις π.χ FORTRAN II, IV, V, 77), ενώ η RPG είχε αρχίσει ήδη να εξαφανίζεται. Στην εποχή του δομημένου προγραμματισμού υπήρχαν οι γλώσσες προγραμματισμού ALGOL 4, PASCAL, ADA, BASIC, COBOL 74, και FORTRAN 77. Από αυτές η PASCAL κατέληξε στη TURBO PASCAL, η ADA σαν επίσημη γλώσσα του NATO, η BASIC σε διάφορες εκδόσεις των Home computers της εποχής εκείνης.

Στις 10 Νοεμβρίου 1983 η Microsoft ανακοίνωσε τα Microsoft Windows, μια επέκταση του λειτουργικού συστήματος MS-DOS, το οποίο παρείχε γραφικό περιβάλλον στους χρήστες των PC. Στην συνέχεια ακολούθησαν οι εκδόσεις Windows 1.0 (1985), 2.0 (1987), 3.0 (1990), 3.11 (1993), NT (1993), 95 (1995), NT4 (1996), 98 (1998), ME (2000), 2000 (2000), XP (2001), 2003 (2003). Στο διάστημα 1990 – 2000 έγινε η επανάσταση στο προγραμματισμό με την εισαγωγή των visual programming εργαλείων και την υποστήριξη από την Visual Basic της Microsoft.

Η COBOL δεν τα πήγαινε γενικά καλά με τους μικροϋπολογιστές από την δεκαετία του 80. Αν και προσπάθησε στη δεκαετία του 90 με τη VISUAL COBOL σήμερα στη πρώτη δεκαετία του 2000 δεν μπορεί κανείς να ισχυριστεί ότι υπάρχει η γλώσσα αυτή. Η FORTRAN μεταλλάχτηκε από FORTRAN 77 στην FORTRAN 90 και στην συνέχεια στην VISUAL FORTRAN ++, κύρια με την συνεργασία της MICROSOFT και της DIGITAL. Είναι όμως τελικά μια δύσκολη γλώσσα, δεν τηρεί ορθά της αρχές του Object Oriented Programming και τελικά αφού κυριάρχησε για δεκαετίες έκλεισε τον κύκλο της αφήνοντας την θέση της σε νεότερες γλώσσες και σύγχρονα λογισμικά εργαλεία. Ωστόσο διδάσκεται ακόμα σε πολλές πολυτεχνικές κυρίως σχολές στη γη ενώ υπάρχουν πολλές βιβλιοθήκες γραμμένες σε FORTRAN. Η γλώσσα προγραμματισμού Visual Basic αναπτύχθηκε χωρίς την υποστήριξη object oriented αρχιτεκτονικής. Είχε σαν κύριο στόχο την παροχή ευκολιών στον προγραμματιστή και τον γρήγορο προγραμματισμό. Η καλύτερη και πιο δημοφιλής εφαρμογή της Visual Basic εμφανίζεται μέσω του προϊόντος Microsoft Access, το οποίο πρωτοεμφανίστηκε το 1992.Η ADA έγινε VISUAL ADA, την οποία όμως γνωρίζουν μόνο οι ιεροφάντες του προγραμματισμού της Εθνικής Άμυνας... Η TURBO PASCAL της BORLAND κατέληξε τελικώς στη DELPHI η οποία όμως δεν συμβιβάστηκε με την αντικειμενοστρεφή τεχνολογία. Ωστόσο η DELPHI αφήνει καλές εντυπώσεις και σήμερα ακόμα και έχει σωστές επαφές με σχεσιακές βάσεις δεδομένων.

Συμπερασματικά οι παραδοσιακές γλώσσες προγραμματισμού υστερούσαν σε διάφορα επίπεδα. Όλες όμως υστερούσαν στην ορθή εφαρμογή

της αντικειμενοστραφής τεχνολογίας, γεγονός αποφασιστικής σημασίας για την εξαφάνιση τους.

1.2 ΓΛΩΣΣΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΟΥΣ ΤΕΧΝΟΛΟΓΙΑΣ

Από τη δεκαετία του 60 ήταν φανερή η έλλειψη σχεδιαστικού εργαλείου ευκολιών και ανάπτυξης προγραμμάτων σε low-level και καλή επαφή με το Master Control Program (MCP). Και βέβαια οι γλώσσες της εποχής εκείνης όπως η FORTRAN δεν ήταν καθόλου ευκίνητες για δόμηση λειτουργικών συστημάτων. Ωστόσο το 1965 στο Cambridge έγινε μια πρώτη απόπειρα με τη γλώσσα BCPL με κύριο στόχο την αντικατάσταση της ASSEMBLY.

Η BCPL δημιούργησε την γλώσσα B από όπου δομήθηκε το κλασσικό λειτουργικό σύστημα UNIX στις ιστορικές μηχανές PDP-7/11 και της DEC. Όμως η γλώσσα B δεν είχε I/O υποσύστημα, έλλειψη που θα γενικευτεί τα υπόλοιπα χρόνια και θα είναι ένα από τα βασικότερα ελαττώματα του μοντέρνου προγραμματισμού των ημερών μας. Μάλιστα ο Dennis Ritchie το 1972 έβγαλε από τη γλώσσα B τη γλώσσα C από όπου προήρθε η ANSI Standard C που παρέμεινε στενός συνεργάτης του UNIX.

Στη δεκαετία του 80 η MICROSOFT έκανε ανεξάρτητη τη C από το UNIX με την MS-C. Η MS-C αλλά και η C με UNIX ήταν εύκολη γλώσσα με μικρό πλήθος εντολών και μπορούσε να την μάθει κάποιος σχετικά εύκολα. Υστερούσε βέβαια σε πολλά θέματα, τα οποία σε παλαιότερες γλώσσες προγραμματισμού (π.χ. PASCAL, ADA, FORTRAN) ήταν απλές δυνατότητες ρουτίνας (π.χ. αυτόματες μετατροπές τύπων δεδομένων, τελεστές του τύπου a^x, ταχύτητες σε binary αρχεία, κτλ). Οπωσδήποτε η γλώσσα C δεν ήταν αυτή που άξιζε, απλά θα είναι ο γεννήτορας αυτών που πραγματικά θα αξίζουν, όπως η C++, JAVA, C#, κτλ.

Η εξέλιξη της C στη C++ και η τυποποίηση αυτής από το ινστιτούτο ANSI θεμελίωσε τις αρχές της αντικειμενοστρεφούς τεχνολογίας ακολουθώντας τον αντικειμενοστρεφή προγραμματισμό που παράλληλα αναπτυσσόταν μέσα στη δεκαετία του 90,(πχ UML). Η C++ είχε ισχυρή τυποποίηση, νέα ρεύματα input/output(iostream) αλλά πάνω από όλα θεμελίωσε τη χρήση της κλάσης σαν αυτόνομο προγραμματιστικό ολόνιο. Μάλιστα από το 1993 και μετά η MICROSOFT βασιζόμενη στις εκδόσεις των **studios** υποστήριζε **την MS VISUAL C++** η οποία κατά κανόνα επεκράτησε. Διακρίνεται σε άριστη συνεργασία με RDBMS, ορθή λειτουργία σε LAN(Local Area Network), Web και άριστη συνεργασία με διαδικτυακό προγραμματισμό, (π.χ JAVA, C#, XML κτλ).

Συμπερασματικά η αντικειμενοστρεφής τεχνολογία άρχισε με τη C++/VISUAL C++ που πήρε τη σκυτάλη από την VISUAL FORTRAN. Σήμερα θεωρείται το αλφάβητο του προγραμματισμού και κανονικά κάθε μηχανικός λογισμικού θα πρέπει άριστα να τη γνωρίζει. Ωστόσο από τη δεκαετία του 90 το μέλλον είχε αρχίσει να επικεντρώνεται στο Διαδίκτυο και ήταν φανερό πως έπρεπε να αναπτυχθούν νέα εργαλεία και γλώσσες προσαρμοσμένα για αυτό.

1.3 ΔΙΑΔΙΚΤΥΑΚΕΣ ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Οι νεότερες εκδόσεις του λειτουργικού **Microsoft Windows**, εκμεταλευόμενες τη ραγδαία εξέλιξη και επανάσταση του **Internet** και του

World Wide Web (WWW) στήριξαν με εργαλεία και γλώσσες τον διαδικτυακό προγραμματισμό στα ακόλουθα χαρακτηριστικά:

- Μεταφερσιμότητα (portability) κατά κανόνα αντιστρόφως ανάλογη του αναμενόμενου υπολογιστικού χρόνου,
- Αντικειμενοστραφής προγραμματισμός κύρια με κλάσεις και αντικείμενα,
- Εύκολη συντήρηση, επέκταση και διαχρονικότητα.

Το 1991 οι ερευνητές της SUN J. Gosling, E. Frank, M. Sheridan, C. Warth δημιούργησαν τη γλώσσα OAK για προγραμματισμό ηλεκτρονικών συσκευών σπιτιού (φούρνοι μικροκυμάτων, τηλεχειριστήρια…). Το 1995 η OAK μετονομάστηκε σε JAVA με λειτουργία σε διαφορετικά περιβάλλοντα, υψηλή απόδοση σε Internet και σε Multimedia εφαρμογές. Παράλληλα το 1990 ο Tim Berners-Lee, στο CERN, επέβαλε μια τάξη στο χάος που κυριαρχούσε στο Διαδίκτυο δημιουργώντας ένα απλό σύστημα κωδικοποίησης των πληροφοριών του σε **Υπερκείμενα (Hypertext).** Έτσι γεννήθηκε η ειδική γλώσσα περιγραφής υπερκείμενων **HyperText Markup Language (HTML**) καθώς και το πρωτόκολλο μεταφοράς δεδομένων **HyperText Transport Protocol (HTTP).** Με τον τρόπο αυτό υλοποιήθηκε η οργάνωση και η μεταφορά για μια σημαντική περιοχή του Internet γνωστή όπως είδαμε με το όνομα Web. Η διαχείριση των ιστοσελίδων, που είναι κατά κανόνα HTML σελίδες, γίνεται με ειδικά προγράμματα περιηγητές (browsers). Έτσι υλοποιείται το τμήμα του Web programming και ειδικότερα σημαντικά κομμάτια αυτού όπως:

- Η τυποποίηση στην οργάνωση κειμένου,
- Ο ορισμός υπερ-συνδέσεων προς άλλα αντικείμενα του διαδικτύου,
- Η δυνατότητα του " Διαβάζω – Βλέπω – Ακούω " (multimedia),
- Η δυνατότητα ενσωμάτωσης εκτελέσιμων προγραμμάτων (μεταφερσιμότητα, ασφάλεια)

Ωστόσο η JAVA δεν κατάφερε να καλύψει βασικές αδυναμίες. Η πρώτη από αυτές ήταν η σχεδόν αδυναμία της ανάμιξης της με πολλές άλλες γνωστές γλώσσες προγραμματισμού (mixed language programming). Βέβαια η ανάμειξη πολλών γλωσσών προγραμματισμού απαιτεί την ύπαρξη μεγάλων κατανεμημένων προγραμματιστικών συστημάτων (Huge distributed software systems). Η JAVA υποστηρίζεται στις νεότερες εκδόσεις των Microsoft Windows μέσω του Virtual Java Machine της SUN. Η Microsoft έχει αναπτύξει το δικό της Virtual Machine για την JAVA το οποίο όμως δεν παρέχει την ίδια υποστήριξη και αξιοπιστία. Το Virtual Machine της Microsoft αναπτύχθηκε για να υποστηρίξει την Visual J++ γλώσσα η οποία ακολούθησε τις προδιαγραφές του JDK μέχρι την έκδοση 1.2.1. Με την εισαγωγή όμως της πλατφόρμας .NET η J++ αντικαταστάθηκε από την J# (java syntax language) και την C# οι οποίες λειτουργούν με ανεξάρτητο περιβάλλον runtime από το VM.Ο δημιουργός της C# είναι ο Anders Hejlsberg, ένας από τους κορυφαίους προγραμματιστές με πληθώρα αξιοσημείωτων επιτευγμάτων στο ενεργητικό του.

Η C# είναι απευθείας συσχετιζόμενη με τη C, C++ και Java, και αυτό δεν είναι τυχαίο καθώς αυτές είναι οι τρεις ποιο διαδεδομένες και χρησιμοποιήσιμες γλώσσες προγραμματισμού στον κόσμο. Για αυτό η σύνταξη της C# έγινε με κατανοητή υποδομή παρεμφερής της C, C++ και Java έτσι ώστε να προσφέρει έναν βατό δρόμο μετανάστευσης από αυτές προς εκείνη. Τελικώς η C# εστίασε τις ελλείψεις που προϋπήρχαν στην γλώσσα προγραμματισμού Java και με συγκεκριμένες βελτιώσεις και καινοτομίες έλυσε τα προβλήματα αυτά.

Συνοψίζοντας αξίζει να σημειωθεί πως η C# προσπαθεί να συνθέσει τη C++ και τη JAVA σε αντίθεση με τη JAVA που ήταν απλά θυγατέρα της C++, γεγονός που καθιστά τη C# "συνδυαστικό" αντικαταστάτη των δυο αυτών σύγχρονων γλωσσών προγραμματισμού.

1.4 ΝΟΡΜΕΣ ΠΡΟΓΡΑΜΜΑΤΩΝ

Στα σύγχρονα πληροφοριακά συστήματα το προϊόν μιας μηχανογράφησης είναι το πρόγραμμα το οποίο μπορεί να είναι:

- Αστερισμός προγραμμάτων (Package),
- Ολοκληρωμένο Πληροφοριακό Σύστημα
- Ένα απλό *.exe/com/bat αρχείο
- Ένα δυναμικό αρχείο που ανοίγει με browser.

Η διάταξη είναι 1>2>3>4, όπου και μια απλή ιστοσελίδα θεωρείται πρόγραμμα… Ένα τέτοιο πρόγραμμα για να θεωρείται αριστοποιημένο πρέπει να ικανοποιεί τέσσερις άριστες νόρμες:

1. Ελαχιστοποίηση Υπολογιστικού Χρόνου
2. Ελαχιστοποίηση αναγκαίας Κύριας Μνήμης
3. Ελαχιστοποίηση Χρόνου Προσπέλασης
4. Μεγιστοποίηση Ευκολίας ανάπτυξης, επέκτασης, διορθώσεων και εξελικτικής προσαρμογής.

ΓΛΩΣΣΕΣ	1	2	3	4	ΝΟΡΜΕΣ
FORTRAN	2	2	3	1	**8**
ADA	3	2	3	1	**9**
PASCAL	2	2	3	1	**8**
C++	3	3	2	2	**10**
JAVA\C#	1	2	2	3	**8**
ΔΥΣΚΟΛΙΑ	**11**	**11**	**13**	**8**	**ΔΗΜΟΤΙΚΟΤΗΤΑ**

[ΚΛΙΜΑΚΑ 1, 2, 3 ΑΡΙΣΤΑ]

Οι γλώσσες JAVA\C# υστερούν μόνο στην ελαχιστοποίηση υπολογιστικού χρόνου μια που τρέχουν μέσα από διερμηνέα (interpreter) αν και η μεν JAVA διαθέτει JIT (Just In Time compilers) για ορισμένα περιβάλλοντα, η δε C# παράγει μέσα από το .NET exe πρόγραμμα αυξάνοντας έτσι την απόδοση στην ελαχιστοποίηση του υπολογιστικού χρόνου και οπωσδήποτε σε αυτήν την περίπτωση ελαττώνοντας παράλληλα την 4η νόρμα της εξελικτικής προσαρμογής.

1.5 ΙΔΙΟΤΗΤΕΣ ΔΙΑΔΙΚΤΥΑΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Οι ακόλουθες ιδιότητες αναφέρονται στα βασικά εργαλεία του Διαδικτυακού προγραμματισμού όπως η JAVA, η C#, μη αποκλειόμενης και της C++:

- Οικεία

Η JAVA είναι απόγονος της C++ με στοιχεία από Smalltalk, FORTRAN, Eiffel, ADA, LISP, MESA και Cedar. Είναι οικεία σε όλους τους προγραμματιστές πλην εκείνων σε COBOL, RPG. Το 70% της JAVA ταυτίζεται με την VISUAL C++. Από την άλλη μεριά η C# αποτέλεσε συνδυασμό της JAVA, C, C++. Κατάφερε να λύσει πολλά προβλήματα των τριών παραπάνω καινοτομώντας.

- Απλά

Η JAVA είναι απλή με ελάχιστες εντολές και με αυτοματισμούς σε διαχείριση μνήμης. Είναι απελευθερωμένη από πολλαπλή κληρονομικότητα και από pointers και άσκοπους τελεστές πράγμα που την κάνει τρωτή σε σχέση με την C# . Η C# κρατάει τα θετικά στοιχεία της JAVA και καλύπτει τα κενά και τις αδυναμίες της.

- Κατανεμημένα

Τόσο η JAVA όσο και η C# προσπελαύνουν αντικείμενα από διαφορετικές κορυφές του διαδικτύου. Επιπλέον είναι πολύ εύκολες στην δημιουργία απλών προγραμμάτων σε επίπεδο server για εξυπηρέτηση clients με πληθώρα εργαλείων για Ολοκληρωμένα Πληροφοριακά Συστήματα σε Internet.

- Αντικειμενοστρεφείς

Και οι δυο λειτουργούν με κλάσεις και αντικείμενα. Οι απόγονοι κληρονομούν δεδομένα και μεθόδους από τις υπερκλάσεις τους. Στην κορυφή της ιεραρχίας είναι η υπερκλάση Object.

- Ερμηνευμένες

Η JAVA για την μετατροπή του κώδικα σε εκτελέσιμο αρχείο κάνει χρήση μιας εικονικής μηχανής JAVA (JAVA Virtual Machine). Ειδικότερα ο διερμηνέας της JAVA μετατρέπει τον κώδικα σε ένα ενδιάμεσο αρχείο *.class υλοποιήσιμο σε αλφάβητο byte codes. Το ενδιάμεσο αρχείο μετατρέπεται σε εκτελέσιμο μέσο του JIT (Just In Time compiler). Είναι με λίγα λόγια ένας compiler για JAVA εκτελέσιμος σε συγκεκριμένη CPU. Παράγει εκτελέσιμο κώδικα υψηλής απόδοσης. Το CLR αντίθετα διαχειρίζεται την εκτέλεση του .NET κώδικα. Όταν μεταγλωττίζετε ένα C# πρόγραμμα, το εξερχόμενο από το μεταγλωττιστή αρχείο δεν είναι το εκτελέσιμο. Εν αντιθέσει, είναι ένα αρχείο που περιέχει ένα ειδικό τύπο ψευδοκώδικα που λέγεται ενδιάμεση γλώσσα της Microsoft (Microsoft Intermediate Language - MSIL), η οποία ορίζει ένα μέρος από εύχρηστες καθοδηγήσεις που είναι ανεξάρτητες από τις υπολογιστικές μονάδες του επεξεργαστή (CPU). Κυρίως η MSIL δημιουργεί τη γλώσσα χαμηλού επιπέδου (assembly language). Με βάση τα προλεχθέντα μπορούμε να παρομοιάσουμε το CLR με το Java Visual Machine, όμως τα δύο αυτά δεν είναι ίδια σε δυνατότητες με το CLR να υπερτερεί.

- Ασφαλείς

Τόσο η JAVA όσο και η C# στηρίζονται και οι δυο σε ένα υποσύστημα δικαιωμάτων πρόσβασης χρηστών στο Δίκτυο . Χρησιμοποιούν ένα

υποσύστημα έλεγχου ενδιάμεσου κώδικα (byte codes verifier) που στηρίζεται στην διαχείριση λαθών μνήμης (memory management mistakes) και στην αντικειμενοστρεφή διαχείριση λαθών (object-oriented exception handling).

- Μεταφέρσιμες

  Μεγάλο πλεονέκτημα και για τις δυο αποτελεί το γεγονός ότι είναι και οι δυο αυτό που θα αποκαλούσαμε γλώσσες ανοικτής αρχιτεκτονικής. Έχουν την δυνατότητα να είναι μεταφέρσιμες σε κάθε σύστημα. Πράγμα που γίνεται εύκολα υλοποιήσιμο για την μεν JAVA μέσω πληθώρας Διερμηνέων για κάθε είδους κορυφής του διαδικτύου, ενώ για την C# με την εγκατάσταση του .NET Framework στα νεότερα στοιχεία.

- Υψηλής απόδοσης

  Τόσο η Java όσο και η C# υστερούν σε ταχύτητα εκτέλεσης του κώδικα από 20 έως 50% η πρώτη και έως 20% η δεύτερη σε σχέση με εκείνη της ταχύτητας εκτέλεσης της C++. Είναι γεγονός ότι προσπαθούν να καλύψουν αυτό το κενό με δύο τρόπους . Ο πρώτος τρόπος είναι τα **πολλαπλά νήματα εκτέλεσης** (**multithreaded**) με υποσύστημα συγχρονισμού για να μην έχουμε πρόωρες εκτελέσεις μεθόδων ή ταυτόχρονες εκτελέσεις ίδιου object. Ο δεύτερος τρόπος είναι με τους **JIT** (**Just In Time Compilers**), όπου, βέβαια, καταστρέφεται η έννοια του ενδιάμεσου δυαδικού κώδικα.

- Υποστήριξη εφαρμογών

  Μεγάλη εφαρμογή και ευελιξία παρουσιάζουν οι JAVA και η C# στην υποστήριξη πολυμεσικών εφαρμογών (multimedia), παίρνοντας υποσυστήματα και συνεργαζόμενα με άλλες κορυφές του Διαδικτύου. Επιπλέον παρέχουν πλήρη υποστήριξη και εύκολη υλοποίηση εφαρμογών βάσεων δεδομένων σε διαδικτυακό περιβάλλον (web-RDBMS) με την χρήση του (JAVA Database Connectivity JDBC) και με την υποστήριξη της τεχνολογίας ADO.NET για την JAVA και την C# αντίστοιχα. Τέλος επιτυγχάνεται πραγματοποίηση σύνθετων οπτικών καταστάσεων και εφαρμογών με την βοήθεια αντίστοιχα των Java Beans και των Graphics και UI (User Interface) της Java και C# αντίστοιχα.

1.6 ΑΡΧΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΟΥΣ ΤΕΧΝΟΛΟΓΙΑΣ

Όπως ήδη έχει τονισθεί, οι αντικειμενοστραφείς γλώσσες προγραμματισμού, JAVA και C# χρησιμοποιούν την Αντικειμενοστραφή Τεχνολογία, την οποία κληρονόμησαν από την C++. Η έννοια Αντικειμενοστρεφής (**object-oriented**) δείχνει έναν μηχανισμό, έναν τρόπο σκέψης στην ανάπτυξη λογισμικού, που είναι στραμμένος σε **Αντικείμενα** (**objects**), τα οποία συγκροτούν τις βασικές δομικές μονάδες της τεχνολογίας αυτής. Το αντικείμενο είναι το βασικό συστατικό της λογισμικής σχεδίασης, είναι η γενίκευση της σύνθετης μεταβλητής και περιλαμβάνει από εγγραφές μέχρι συναρτήσεις και υποπρογράμματα ή διαδικασίες και μεθόδους. Το αντικείμενο έχει τις ακόλουθες βασικές ιδιότητες:

- **Κατάσταση:** Έχει να κάνει με τις τιμές του αντικειμένου σε επίπεδο μεταβλητών μνήμης

- **Συμπεριφορά:** Είναι ο τρόπος που ανταποκρίνεται το αντικείμενο σε κλήσεις από το περιβάλλον του.

- **Ταυτότητα:** Είναι η διάκριση ενός αντικειμένου από όλα τα άλλα που ορίζονται στον ίδιο τύπο δεδομένων.

Τα σύνολα των αντικειμένων που έχουν κοινή **Συμπεριφορά** και δομή, συγκροτούν το πλέον σύγχρονο λογισμικό ολόνιο, την **Κλάση** (Class). Πρόγονος της Κλάσης είναι το type (τύπος) της PASCAL του Δομημένου Προγραμματισμού. Κάθε αντικείμενο ανήκει σε μια Κλάση, της οποίας ορίζει ένα **Στιγμιότυπο** (instance) το οποίο είναι και το μόνο που απασχολεί κύρια μνήμη για καταχώριση των τιμών του.

Το βασικό συστατικό ενός αντικειμένου είναι το **Πεδίο** (field), ενώ το σύνολο των πεδίων του αποτελεί την **Κατάσταση** του αντικειμένου. Επίσης, η **Συμπεριφορά** ενός αντικειμένου εκφράζεται με το σύνολο των ενεργών συστατικών λογισμικού (συνάρτηση, διαδικασία, υποπρόγραμμα κλπ.), που ονομάζονται **Μέθοδοι**.

Η έννοια της **Ενθυλάκωσης** (Encapsulation) ή και **Κελυφοποίησης** ταυτίζεται με την μερική πρόσβαση σε ορισμένα πεδία ή και μεθόδους ενός Αντικειμένου αποκρύπτοντας τα υπόλοιπα συστατικά του (**Information Hiding**) αν και στην κύρια μνήμη υπάρχει το πλήρες Στιγμιότυπο του Αντικειμένου.

Δύο ή και περισσότερες κλάσεις, μπορούν να σχετίζονται μεταξύ τους με την έννοια της **Συσχέτισης** (Association) στα δεδομένα τους ή και στις μεθόδους τους. Μια τέτοια σχέση χαρακτηρίζεται από το Όνομα της Συσχέτισης, το οποίο μάλιστα δεν είναι κατ' ανάγκη το ίδιο από την πρώτη προς την δεύτερη Κλάση από εκείνο της δεύτερης προς την πρώτη. Επίσης, μια Συσχέτιση χαρακτηρίζεται από τις ιδιότητες του **Ρόλου** (Role) και της **Πολλαπλότητας**. Η τελευταία δηλώνει το ελάχιστο και μέγιστο πλήθος των μελών κάθε μέρους της Συσχέτισης, ενώ ο Ρόλος περιορίζεται συνήθως σε κάποιον προτασιακό τόπο που εξηγεί απλά σε φυσική γλώσσα την δραστηριότητα της Συσχέτισης από τη μια Κλάση στην άλλη.

Η **Κληρονομικότητα** (Inheritance) ή **Γενίκευση** (Generalization) είναι αναμφίβολα η σοβαρότερη ιδιότητα στην Αντικειμενοστρεφή Τεχνολογία. Δύο κλάσεις συνδέονται με την ιδιότητα της Κληρονομικότητας, όταν η πρώτη δίνει στη δεύτερη τα χαρακτηριστικά της. Η πρώτη Κλάση ονομάζεται **Κλάση-Πατέρας** ενώ η δεύτερη, **Κλάση-Παιδί**. Όταν ο προγραμματιστής βλέπει από την Κλάση-Πατέρα προς την Κλάση-Παιδί έχουμε την Κληρονομικότητα, διαφορετικά έχουμε τη Γενίκευση:

Κληρονομικότητα	**: Πατέρας**	**➔ Παιδί (ή Εξειδίκευση)**
Γενίκευση	**: Παιδί**	**➔ Πατέρας**

Τα επιπρόσθετα χαρακτηριστικά της Κλάσης-Παιδί εξειδικεύουν την κληρονομικότητα, γι' αυτό θεωρείται σαν μια **Εξειδίκευση** της Κλάσης-Πατέρα. Ο Προγραμματιστής, όταν στο στάδιο σχεδιασμού παρατηρεί την

ύπαρξη τομής σε πεδία ή και μεθόδους μεταξύ δύο ή και περισσοτέρων Κλάσεων, τότε η τομή αυτή συμφέρει να ορισθεί σαν Κλάση-Πατέρας των άλλων Κλάσεων.

Στη Θεωρία Προγραμματισμού, συχνά η Κλάση-Πατέρας ονομάζεται και **Βασική Κλάση**, ενώ η Κλάση-Παιδί, **Παραγόμενη Κλάση**. Στην περίπτωση που μια Κλάση έχει μόνο μια Βασική Κλάση, τότε έχουμε μια **Κληρονομικότητα τάξεως 1** (Inheritance of order 1). Γενικά, όταν μια κλάση έχει χαρακτηριστικά από Κ Βασικές Κλάσεις, έχουμε μια πολυπλοκότητα Κληρονομικότητας ή **Κληρονομικότητα Τάξεως Κ** (Inheritance of order K). **Ευτυχώς για τον Προγραμματιστή της JAVA και C# η Κληρονομικότητα περιορίζεται (απλουστεύεται) στην περίπτωση του Κ=1**. Επίσης το ίδιο ισχύει και σε κάθε είδος αντικειμενοστρεφούς γλώσσας όπως στην Visual J++ και στην γλώσσα της Microsoft, την C#.

Μια επίσης βασική ιδιότητα της Αντικειμενοστρεφούς Φιλοσοφίας είναι το φαινόμενο της **Συναρμολόγησης** (Aggregation), όπου γίνεται σύνθεση συνόλου από απλά μέρη. Τα απλά αυτά μέρη μπορεί να έχουν και κοινά στοιχεία μεταξύ τους ή και να είναι ακόμη και πιστά αντίγραφα μεταξύ τους, ιδιαίτερα σε περιπτώσεις **συστημάτων ανάκτησης** (recovery systems), όπου ο βασικός στόχος είναι η ελαχιστοποίηση της πιθανότητας κατάρρευσης του όλου Συστήματος.

Ένα από τα πιο χρήσιμα εργαλεία για έκφραση των μοντέλων που αντιστοιχούν σε θέματα της πραγματικότητας, είναι η **Αφαίρεση** (Abstraction), η οποία δίνει την δυνατότητα από το σύνολο των λεπτομερειών ενός προγραμματιστικού ολονίου, να επικεντρώσουμε την προσοχή μας σε ορισμένες από αυτές, αφαιρώντας τις υπόλοιπες. Η ίδια η έννοια της Κλάσης αποτελεί την έκφραση της Αφαίρεσης, όπου γίνεται ομαδοποίηση δεδομένων και μεθόδων, έτσι ώστε να έχουμε πιστότερη προβολή των οντοτήτων του πραγματικού κόσμου.

Τέλος, κάθε έννοια της Αντικειμενοστρεφούς Τεχνολογίας απαιτεί την ύπαρξη ενός **πρότυπου συμβολισμού**. Προσπάθειες τυποποίησης του συμβολισμού άρχισαν από το 1990 (Booch, Jacobson, Rumbaugh) και συνεχίσθηκαν από κατασκευαστές εργαλείων σε όλη τη δεκαετία αυτή. Τελικά, έγινε διεθνώς δεκτό το πρότυπο **UML** (Unified Modeling Language), το οποίο υιοθετήθηκε (έκδοση 1999 UML 1.3 και μετά) από τον οργανισμό OMG (Object Management Group: www.omg.org) σαν Βιομηχανικό Πρότυπο Παράστασης Λογισμικού.

Στο σημείο αυτό θα πρέπει να τονισθεί πως ο Αντικειμενοστρεφής Προγραμματισμός δε γεννήθηκε από τη μια μέρα στην άλλη, ούτε είναι η «αποκάλυψη» του Προγραμματισμού, όπως ορισμένοι νεώτεροι Προγραμματιστές νομίζουν! Σχέσεις όπως η Κληρονομικότητα και η Συναρμολόγηση υπήρχαν μερικά και καλυμμένα στον Δομημένο Προγραμματισμό. Για παράδειγμα η παραμετροποίηση μιας διαδικασίας στον παραδοσιακό προγραμματισμό είναι μια μορφή Κληρονομικότητας, όπως είναι και ο τύπος δεδομένων τύπου RECORD της PASCAL με μεταβλητό / παραμετρικό περιεχόμενο (CASE OF). Αλλά και ο ορισμός του TYPE της PASCAL ή του STRUCT της απλής C, δεν ήταν τίποτε άλλο από την έννοια αυτής της ίδιας της Κλάσης χωρίς Μεθόδους όμως. Όμοια. και η έννοια της Αφαίρεσης υπήρχε μονόπλευρα σαν functional Abstraction, σαν μέρη δηλαδή προγράμματος σε εντελώς ανεξάρτητα δεδομένα. Η Κλάση όμως ακολουθεί την

Αφαίρεση και ως προς τα δεδομένα (data abstraction) όπου μαζί με αυτά ενυπάρχουν και οι Μέθοδοι. Αλλά ακόμη και αυτό δεν ήταν άγνωστο, διότι υπήρχε για δεκαετίας στην Τεχνολογία Γνώσης (Knowledge Engineering), όπου data (γνώση) και λειτουργίες (μέθοδοι) εμπλέκονται σε ενιαίο σύνολο. Έχουμε τη γνώμη πως η Αντικειμενοστρεφής Προσέγγιση έγινε πρόσφορη και καταξιώθηκε στον κόσμο των μηχανικών λογισμικού, με την εμφάνιση των Δικτυοκεντρικών Εφαρμογών μεγάλης κλίμακας και υψηλών επιδόσεων, με την οργάνωση και επιστημονική αντιμετώπιση τεχνικών που λίγο ή πολύ είχαν οδηγηθεί οι έμπειροι Αναλυτές παλαιότερων εποχών.

1.7 ΑΣΚΗΣΕΙΣ

1. Αναφέρατε τους τομείς για τους οποίους ενδείκνυται η χρήση των γλωσσών Cobol, Fortran και RPG καθώς και τα μειονεκτήματα που ΄χει η καθεμία από αυτές.

2. Αναφέρατε τις βασικές νόρμες ενός προγράμματος ως προς τις οποίες θεωρείται αριστοποιημένο καθώς και τις νόρμες στις οποίες υπερτερούν και υστερούν αντίστοιχα η Java και η C#:

3. Ποιες οι ιδιότητες του Διαδικτυακού προγραμματισμού;

4. Να αναφερθούν ομοιότητες παραδοσιακού προγραμματισμού με Διαδικτυακό προγραμματισμό.

5. Αναφέρεται σε ποια σημεία παρουσιάζει ελλείψεις η Java και πως καλύπτονται αυτές οι ελλείψεις από τη C#

6. Ποια είναι τα βασικότερα στοιχεία του αντιτκειμενοστρεφούς προγραμματισμού; Αναλύστε τις έννοιες:

    - Κλάση
    - Πεδίο
    - Μέθοδοι
    - Ρόλος
    - Κληρονομικότητα
    - Συσχέτιση
    - Στιγμιότυπο

ΚΕΦΑΛΑΙΟ 2ο

2. ΒΑΣΙΚΕΣ ΑΡΧΕΣ JAVA ΚΑΙ C#

2.1 ΜΕΤΑΓΛΩΤΤΙΣΗ ΠΡΟΓΡΑΜΜΑΤΩΝ

Αναφορικά με την μεταγλώττιση προγραμμάτων, η Java, αντί για linking, αναλαμβάνει μέσα στα πλαίσια του **JVM (java Virtual Machine)**, ο φορτωτής κλάσεων με ελεγκτή τον **Bytecode Verifier**. Τέλος, το **Runtime System** θα αναλάβει την κατασκευή και καταστροφή των object και γενικά την διαχείριση της μνήμης. Σε αντίθεση με τη Java, όπως θα παρουσιάσουμε αναλυτικά στην συνέχεια, ένα πρόγραμμα C# κατά το compilation του δημιουργεί μια **Intermediate γλώσσα** τον ενδιάμεσο κώδικα του προγράμματος, πετυχαίνοντας έτσι, το compilation να γίνεται σε γλώσσα μηχανής κάτι που εξασφαλίζει καλύτερες ταχύτητες κατά την εκτέλεση του προγράμματος. Παράλληλα, υποστηρίζει και αντικείμενα του **.NET**. Τη διαχείριση αυτής της ενδιάμεσης γλώσσας, αλλά και την παραγωγή του Πηγαίου κώδικα – στο .NET - έχει αναλάβει το **CLS (Common Language Specification).** Τα παραπάνω μαζί με άλλα στοιχεία αποτελούν το σύνολο των Μεταδεδομένων που περιέχουν πληροφορίες για μεθόδους, τύπους και κλάσεις και αποθηκεύονται στους φακέλους **Assembly**. Τέλος, ο **Garbage Collector** διαχειρίζεται την μνήμη του συστήματος.

2.1.1 ΤΙ ΕΙΝΑΙ ΤΟ .NET FRAMEWORK

Το .NET Framework καθορίζει ένα περιβάλλον που θα παρέχει την ανάπτυξη και εκτέλεση υψηλών-κατανεμημένων βασισμένων σε components εφαρμογών (highly-distributed component-based applications). Ενεργοποιεί τη δυνατότητα να λειτουργούν **διαφορετικές γλώσσες προγραμματισμού μαζί και παρέχει ασφάλεια**, **προγραμματιστική ευχρηστία**, και ένα κοινό προγραμματιστικό μοντέλο για τη πλατφόρμα των Windows.

Μέσα από τη σύνδεση της C#, το .NET Framework προσφέρει δύο ενδιαφέρουσες έννοιες. Η πρώτη είναι η **(Common Language Runtime - CLR**). Αυτό είναι το σύστημα που διευθύνει την εκτέλεση του προγράμματος. Μαζί με τα επιπρόσθετα προτερήματα, το CLR είναι μέρος του .NET Framework το οποίο βοηθά τα προγράμματα να είναι εύχρηστα, ενισχύει τη δυνατότητα ανάμιξης πολλών γλωσσών προγραμματισμού και παρέχει ασφάλεια.

Η δεύτερη έννοια είναι η **.NET βιβλιοθήκη κλάσεων (class library)**. Αυτή η βιβλιοθήκη δίνει στα προγράμματα πρόσβαση στο περιβάλλον χρόνου εκτέλεσης (runtime environment). Για παράδειγμα, εάν θέλετε να εκπληρώσετε I/O (Είσοδος / Έξοδος), όπως να εμφανισθεί ένα μήνυμα στην οθόνη, θα χρησιμοποιήσετε τη βιβλιοθήκη κλάσεων για να το πραγματοποιήσετε. Εφόσον το πρόγραμμα περιορίζει από μόνο του τα καθορισμένα features μέσω της βιβλιοθήκης κλάσεων του .NET, αυτά μπορούν να τρέχουν παντού όπου υποστηρίζεται το .NET σύστημα χρόνου εκτέλεσης. Από τη στιγμή που η C# χρησιμοποιεί αυτόματα τη βιβλιοθήκη κλάσεων του .NET είναι αναγνωρίσιμη σε όλα τα περιβάλλοντα .NET.

2.1.2 ΠΩΣ ΔΟΥΛΕΥΕΙ Η ΚΟΙΝΗ ΓΛΩΣΣΑ ΧΡΟΝΟΥ ΕΚΤΕΛΕΣΗΣ

Το CLR διαχειρίζεται την εκτέλεση του .NET κώδικα. Όταν μεταγλωττίζετε ένα C# πρόγραμμα, το εξερχόμενο από το μεταγλωττιστή αρχείο δεν είναι το εκτελέσιμο. Εν αντιθέσει, είναι ένα αρχείο που περιέχει ένα ειδικό τύπο ψευδοκώδικα που λέγεται **ενδιάμεση γλώσσα της Microsoft (Microsoft Intermediate Language - MSIL)**, η οποία ορίζει ένα μέρος από εύχρηστες καθοδηγήσεις που είναι ανεξάρτητες από τις υπολογιστικές μονάδες του επεξεργαστή (CPU). Κυρίως η MSIL δημιουργεί τη **γλώσσα χαμηλού επιπέδου (assembly language)**.

Η ύπαρξη του CLR έγκειται στο να μετατρέπει το MSIL σε εκτελέσιμο κώδικα όταν το πρόγραμμα εκτελείται. Συνεπώς, οποιοδήποτε πρόγραμμα μεταγλωττίζεται στο MSIL μπορεί να εκτελείται σε κάθε περιβάλλον στο οποίο παρέχεται το CLR. Αυτό είναι το σημείο στο οποίο το .NET Framework πετυχαίνει ευχρηστία σε όλα τα περιβάλλοντα.

Η ενδιάμεση γλώσσα της Microsoft (MSIL) μετατρέπεται σε εκτελέσιμο κώδικα χρησιμοποιώντας τον **JIT μεταγλωττιστή (Just-In-Time)**. Η διαδικασία είναι η εξής: Όταν ένα .NET πρόγραμμα εκτελείται, το CLR ενεργοποιεί τον JIT μεταγλωττιστή. Ο JIT μεταγλωττιστής μετασχηματίζει MSIL σε εκ γενετή κώδικα (native code) πάνω σε βασισμένη διαταγή καθώς κάθε μέρος του προγράμματος το χρειάζεται. Συνεπώς, το C# πρόγραμμα πραγματικά εκτελείται σαν εκ γενετή κώδικα αν και κατά αρχήν μεταγλωττίζεται σε MSIL. Αυτό μεταφράζεται ότι το πρόγραμμα τρέχει τόσο γρήγορα όσο εάν είχε μεταγλωττιστεί σε εκ γενετή κώδικα από την αρχή, αλλά κέρδισε τα προτερήματα ευχρηστίας από το MSIL.

Εν κατακλείδι, το MSIL εξάγει και ένα άλλο πράγμα κατά τη μεταγλώττιση ενός C# προγράμματος: **τα metadata**. Ως metadata χαρακτηρίζονται τα δεδομένα που χρησιμοποιούνται από το πρόγραμμα και επιτρέπουν στο κώδικα να αλληλεπιδρά με διαφορετικό κώδικα. Τα Metadata συμπεριλαμβάνονται στο ίδιο αρχείο όπως το MSIL.

2.1.3 ΔΙΑΧΕΙΡΙΖΟΜΕΝΟΣ ΚΑΙ ΜΗ ΚΩΔΙΚΑΣ

Γενικά, όταν γράφετε ένα C# πρόγραμμα, δημιουργείται αυτό που αποκαλείται **διαχειριζόμενος κώδικας (managed code).** Ο διαχειριζόμενος κώδικας εκτελείται κάτω από τον έλεγχο του CLR με τη μεθοδολογία που παρουσιάστηκε. Από τη στιγμή που εκτελείται κάτω από τον έλεγχο του CLR, ο διαχειριζόμενος κώδικας είναι αντικείμενο στους προκαθορισμένους περιορισμούς και παράγει αρκετά πλεονεκτήματα. Οι περιορισμοί μπορούν εύκολα να περιγραφτούν και να συναντηθούν καθώς προγραμματίζετε σε C#. Οι περιορισμοί αυτοί είναι ότι ο μεταγλωττιστής πρέπει να δημιουργήσει ένα MSIL αρχείο στοχεύοντας για το CLR και για να χρησιμοποιήσει τη βιβλιοθήκη κλάσεων του .NET Framework. Από την άλλη πλευρά τα πλεονεκτήματα είναι αξιοσημείωτα όπως παρέχεται ένας σύγχρονος διαχειριστής μνήμης, διαθέτει την ικανότητα να αναμιγνύει διαφορετικές γλώσσες προγραμματισμού, εξασφαλίζει μεγαλύτερη ασφάλεια, υποστηρίζει έλεγχο για νεότερες εκδόσεις και μια καθαρή οδό για να αλληλεπιδράσουν τα components του λογισμικού.

Από την αντίθετη πλευρά είναι **ο μη διαχειριζόμενος κώδικας (unmanaged code)** ο οποίος δεν εκτελείται κάτω από την έλεγχο του CLR. Συνεπώς όλα τα προγράμματα που υπήρχαν πριν της εκδόσεως του .NET Framework χρησιμοποιούσαν μη διαχειριζόμενο κώδικα. Είναι πιθανό ο διαχειριζόμενος και μη κώδικας να εκτελούνται μαζί, έτσι ώστε η γέννηση διαχειριζόμενου κώδικα από τη C# δεν περιορίζει την δυνατότητα να λειτουργούν σε συνδυασμό με τα προϋπάρχοντα προγράμματα.

2.1.4 Ο ΚΑΘΟΡΙΣΜΟΣ ΤΗΣ ΚΟΙΝΗΣ ΓΛΩΣΣΑΣ

Το **CMS (Common Language Specification)** προσφέρει ανοικτές προδιαγραφές που θα πρέπει να τηρεί οποιαδήποτε γλώσσα προγραμματισμού θέλει να εκτελείται μέσω του CLR και να αποκαλείται .NET γλώσσα προγραμματισμού. Το CMS περιέχει το **CTS (Common Type System)** το οποίο αφορά κοινούς τύπους δεδομένων για όλες τις .NET γλώσσες προγραμματισμού ώστε να είναι εφικτή η δημιουργία εφαρμογών από source κώδικα διαφορετικών .NET γλωσσών προγραμματισμού.

2.1.5 ΣΥΜΠΕΡΑΣΜΑΤΑ ΜΕΤΑΓΛΩΤΤΙΣΗΣ

Όλα τα παραπάνω συνοψίζονται γραφικά στα δυο ακόλουθα σχήματα. Αρχικά γίνεται μεταγλώττιση του πηγαίου προγράμματος JAVA (έστω το X.java) με τη βοήθεια του Compiler της JAVA, του **javac** με την απλή εντολή:

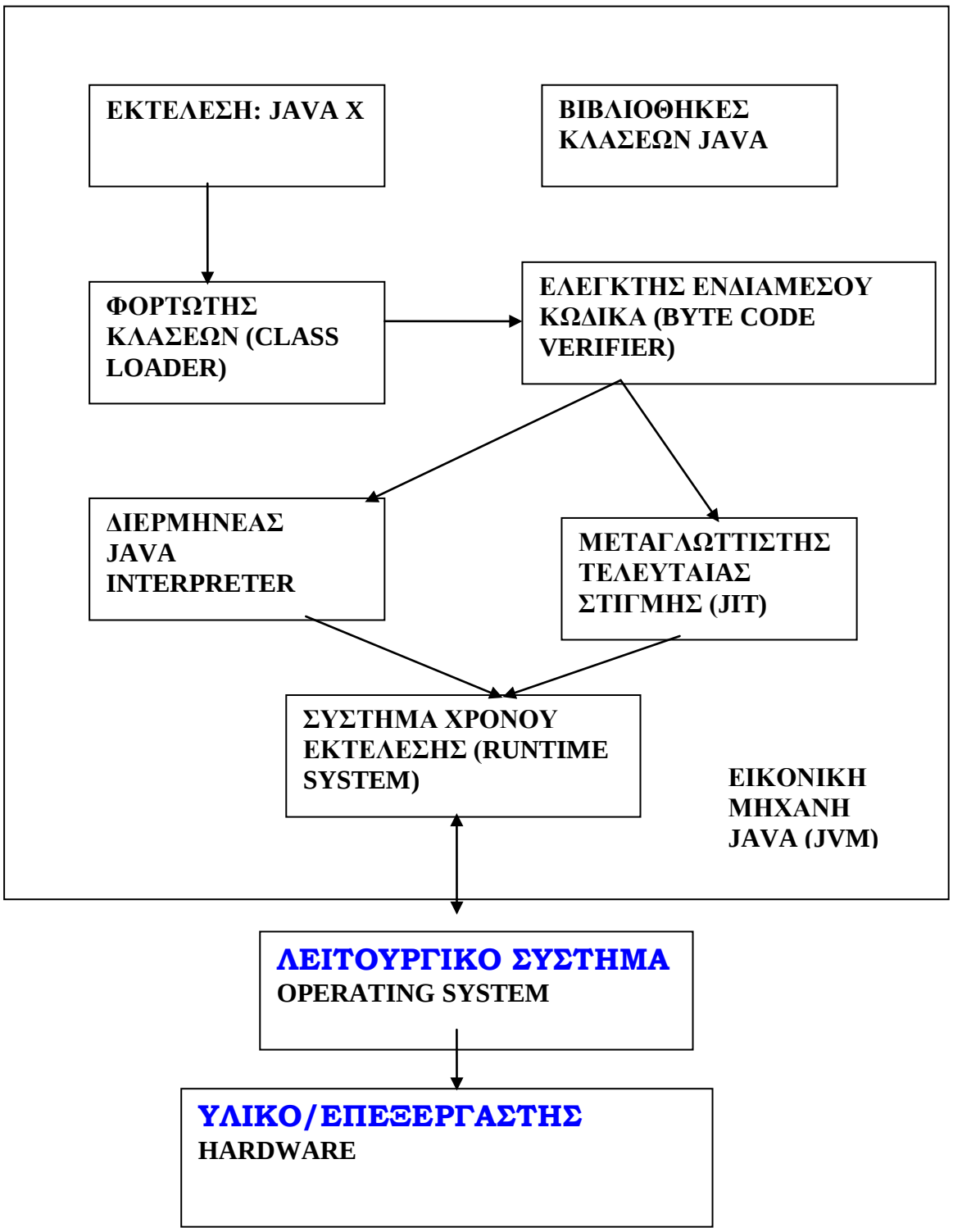
```
javac X.java
```

Αν δεν υπάρχουν μεταφραστικά λάθη, παράγεται ο ενδιάμεσος κώδικας σε bytecode σε ένα ή και περισσότερα αρχεία (έστω το X.class), τα οποία μπορούν να χρησιμοποιηθούν τοπικά ή και να μεταφερθούν σε άλλες κορυφές του Δικτύου.

Όταν δημιουργηθεί ο ενδιάμεσος κώδικας (bytecode), γίνεται η εκτέλεση αυτού από την **Εικονική Μηχανή JAVA** (**JVM**: Java Virtual Machine) και όχι μόνον από τον **Διερμηνέα** που είναι ένα απλό μέρος του JVM. Η εκτέλεση αρχίζει με την απλή διαταγή:

```
java X
```

Αμέσως αρχίζει η αντίστοιχη εργασία του Linking στις παραδοσιακές γλώσσες προγραμματισμού, δηλαδή η αναζήτηση και ενσωμάτωση επιπρόσθετου ενδιάμεσου κώδικα που αναφέρεται σε κλάσεις που χρησιμοποιούνται από το πρόγραμμα.

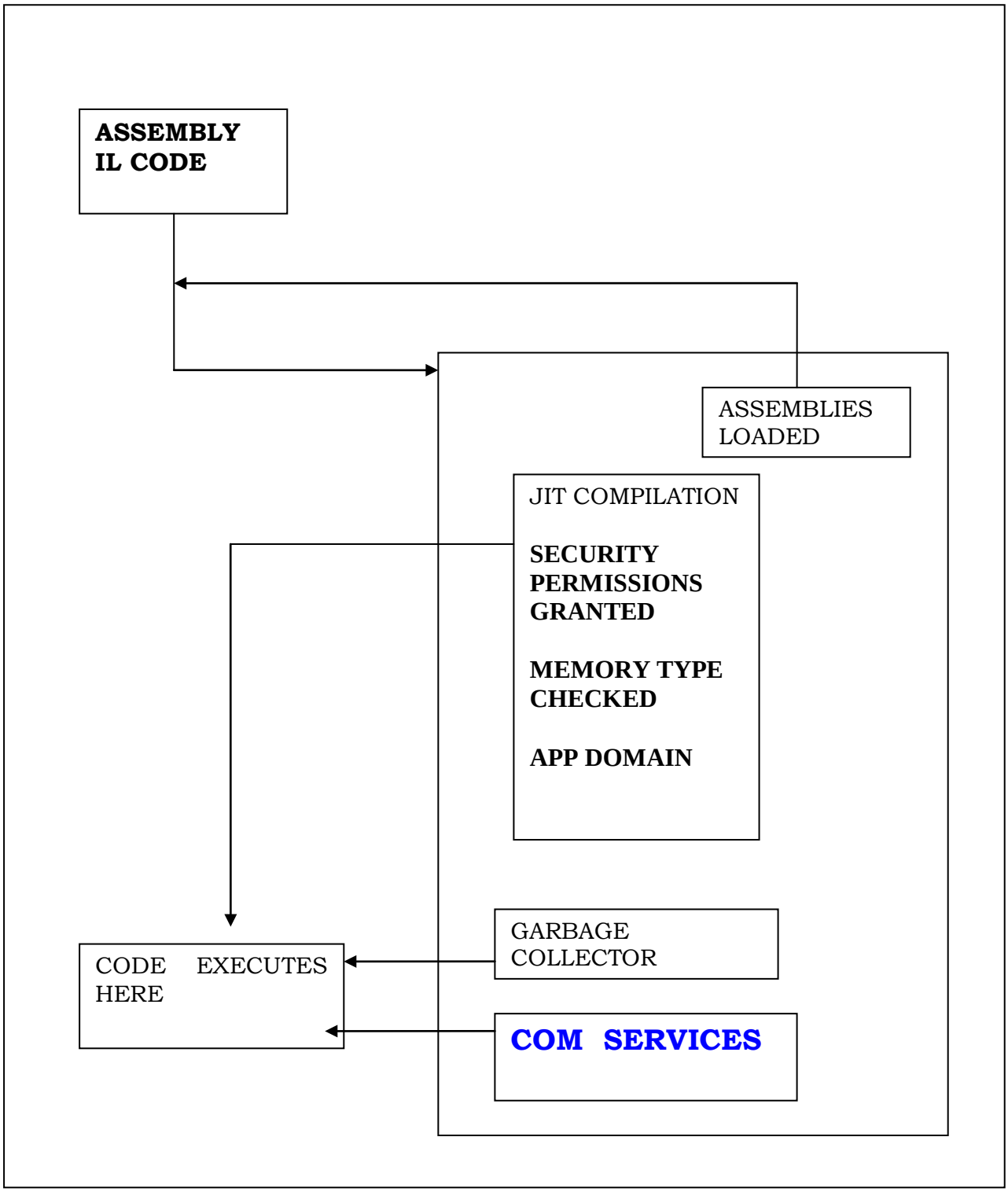


Τελευταίας Στιγμής (JIT: Just-In-Time Compiler) την κανονική εκτέλεση του προγράμματος. Σε κάθε περίπτωση το **Σύστημα Χρόνου Εκτέλεσης** (Runtime System) του JVM θα αναλάβει την κατασκευή και την καταστροφή αντικειμένων και γενικά την πλήρη διαχείριση της μνήμης σε πλήρη αλληλεπίδραση και συνεργασία με το Λειτουργικό Σύστημα, όπου ο Χρήστης επικοινωνεί με το Υλικό της μηχανής του.

Από την άλλη μεριά, όπως είδαμε και παραπάνω, η μεταγλώττιση στη C# γίνεται πολύ εύκολα, αφού δεν χρειάζεται ένα ξεχωριστό προγραμματιστικό εργαλείο, για να μεταγλωττισθούν οι εφαρμογές σε IL (Ενδιάμεση Γλώσσα). Ανοίγοντας ένα παράθυρο εντολών, στον υποκατάλογο που έχει αποθηκευτεί το MyFile.cs, δίνουμε την ακόλουθη εντολή:

csc MyFile.cs

Το MyFile.cs μεταγλωττίστηκε και συνδέθηκε με το MyFile.exe. Αν το πρόγραμμα είναι χωρίς λάθη, η μεταγλώττιση τελειώνει και απλώς πληκτρολογούμε MyFile στην γραμμή εντολών για να τρέξει η εφαρμογή μας



2.2 ΒΙΒΛΙΟΘΗΚΕΣ ΚΛΑΣΕΩΝ-ΕΚΤΕΛΕΣΗ

2.2.1 ΜΟΡΦΗ ΑΥΤΟΝΟΜΩΝ ΕΦΑΡΜΟΓΩΝ

Στη συνέχεια θα δοθεί ένα απλό παράδειγμα αυτόνομου προγράμματος JAVA και το αντίστοιχό του σε C#. Ξεκινώντας από την περίπτωση της Java, όλα αρχίζουν από έναν απλό Editor (π.χ., το Notepad των MS-Windows) όπου ο προγραμματιστής, έστω ότι πληκτρολόγησε το παρακάτω απλό παράδειγμα Java001, το οποίο έσωσε στο αρχείο Java001.java.

```
/*  Πρόγραμμα Java001
    Ζητείται από το Χρήστη ορθό Παρασύνθημα.
    Ορθό Παρασύνθημα είναι εδώ κάθε παράθεση
    χαρακτήρων που αρχίζει με τη συμβολοσειρά JAVA*/
// Η εκτέλεση θα γίνει σα java Java001
class Java001
    {
    public static void main(String args[])
        {
        System.out.print("YOU GAVE THE PASSWORD ===> ");
        System.out.println(args[0]);
        if (args[0].startsWith("JAVA"))
            System.out.print("O.K. you gave correct" + "
    Password ... ");
        else
            System.out.print("I  do not know you ! ");
        }
    }
```

Στη συνέχεια παρατίθεται το ίδιο πρόγραμμα, γραμμένο σε κώδικα C#.

```
using System;

namespace Password
{
    class CPassword
    {
        [STAThread]
        static void Main(string[] args)
        {
            // TODO: Add code to start application here
            Console.Write("You Gave the Password===>");
            Console.WriteLine(args[0]);
            if (args[0].StartsWith("JAVA"))
                Console.Write("Ok, You Gave Correct Password");
            else
                Console.Write("I do not Know You");}}}
```

Το πρώτο σημείο άξιο παρατήρησης, είναι ότι **στη JAVA, συμφέρει τον προγραμματιστή να σώνει τον πηγαίο κώδικα με όνομα αρχείου που να ταυτίζεται με το όνομα της κλάσης που περιέχει τη Μέθοδο main**. Αυτό σημαίνει πως στο παράδειγμά μας, θα πρέπει το αρχείο του πηγαίου προγράμματος να φέρει το όνομα Java001.java, δηλαδή το όνομα της Κλάσης με όνομα επέκτασης βέβαια το "java", το οποίο είναι υποχρεωτικό για τον Μεταγλωττιστή της JAVA αλλά και στη C#, αυτό είναι αναγκαίο, όπως άλλωστε, φαίνεται και στο παράδειγμα. εδώ, **ο χρήστης μπορεί να σώσει τον πηγαίο του κώδικα με όποιο όνομα αυτός επιθυμεί ίδιο με το όνομα της κύριας κλάσης του προγράμματός** του και στη συνέχεια να μεταγλωττίσει το πρόγραμμά του.

Αν δεν υπάρχουν λάθη και στις δυο περιπτώσεις, θα εκτελεστούν τα αντίστοιχα ενδιάμεσα αρχεία, όπως είδαμε σε προηγούμενη παράγραφο. Παράλληλα, οι παράμετροι στη **γραμμή εκτέλεσης** (Command Line) είναι γνωστοί στα προγράμματα που περνούν με τη βοήθεια των παραμέτρων της βασικής Μεθόδου main. Στην περίπτωσή μας η εκτέλεση του προγράμματος θα δώσει την ακόλουθη έξοδο:

```
YOU GAVE THE PASSWORD ===> JAVAPassword
O.K. you gave correct Password
```

διότι ο Χρήστης έδωσε παρασύνθημα ορθό (αρχίζει με τη λέξη "JAVA"), το οποίο πέρασε σαν είσοδο (Input) στο πρόγραμμα από την γραμμή εκτέλεσης.
Οπωσδήποτε μια πλησιέστερη ματιά στον πηγαίο κώδικα του παραδείγματος είναι αναγκαία για να μην έχει ο Αναγνώστης μεγάλα γνωσιολογικά κενά στη συνέχεια:

- Οι διερμηνείς των JAVA-C# **ξεχωρίζουν τα μικρά από τα κεφαλαία γράμματα**. Αυτό σημαίνει πως τα ονόματα Java001 και java001 και αντίστοιχα Password, password είναι τελείως διαφορετικά, γεγονός που ο Χρήστης πρέπει να σταθμίσει ιδιαίτερα.
- Το κύριο πρόγραμμα σε JAVA ονομάζεται αναγκαστικά **main** όπως και στη C/C++, μόνο που στη JAVA είναι και αυτό ένα στοιχείο (Μέθοδος) μιας κλάσης, της κλάσης Java001 στο παράδειγμά μας. Αντίθετα στη C# η main μέθοδος είναι τελείως διαφορετική. Πρώτον, γράφεται με Μ κεφαλαίο. Δεύτερον, στην Java η main μέθοδος έχει την παρακάτω σύνταξη

```
public static void main(String[] args)
```

Απεναντίας η μέθοδος Main στη C# μπορεί να επιστρέψει int ή void και δεν είναι αναγκαίο να παίρνει string arguments (ζητούμενα). Άρα, έχουμε τις ακόλουθες διαφορετικές εκφράσεις της Main:

```
static void Main()
static void Main(String[] args)
static int Main()
static int Main(String[] args)
```

Οι λεπτομέρειες που πρέπει να τονίσουμε είναι ότι

1. Ο public προσαρμογέας πρόσβασης δεν είναι απαιτούμενος στην μέθοδο Main της C#. Ο δείκτης εισόδου της εφαρμογής είναι πάντοτε προσιτός του χρόνου εκτέλεσης (runtime) αδιάφορος όμως τις δηλώσεις προσβασιμότητας.
2. Το int λειτουργεί σαν κατάληψη στατικού κώδικα (termination static code) το οποίο επιστρέφει στο εκτελέσιμο περιβάλλον την περάτωση του προγράμματος. Η ίδια λειτουργία περατώνεται ισάξια στη Java χρησιμοποιώντας την Runtime.getRuntime.exit(int) ή System.exit(int).

- Κάθε λογισμική παράγραφος αρχίζει με το σύμβολο { **(curly brace)** και κλείνει επίσης με τον χαρακτήρα }, καθώς επίσης κάθε σύνθετη εντολή, Μέθοδος, διαδικασία, κλπ., ενώ ο **διαχωριστής εντολών** είναι κατά τα γνωστά (για τους Χρήστες της C++) ο χαρακτήρας **; (semicolon).**
- Οι παρατηρήσεις (**comments**) μπορούν να περιλαμβάνουν πολλές γραμμές, οπότε ακολουθούν τον συντακτικό τύπο:

```
/* κείμενο παρατήρησης σε γραμμές */
```

ή και να εκφράζουν μια μικρή παρατήρηση μιας γραμμής ή και από το σημείο που σημειώνονται μέχρι το τέλος της παρούσης γραμμής, σύμφωνα με τον συντακτικό τύπο:

```
// παρατήρηση μιας γραμμής
```

Υπενθυμίζεται πως οι παρατηρήσεις αφορούν αποκλειστικά το στάδιο του πηγαίου προγράμματος και στην ουσία δεν υπάρχουν στα επόμενα στάδια μετάφρασης / εκτέλεσης.

Η συνάρτηση **Write** (ή **WriteLine** για εκτύπωση και αλλαγή γραμμής), είναι απλά μια Μέθοδος του αντικειμένου **Console** που συνδέεται με έξοδο στην κονσόλα (όπως στο stdout της C/C++), το οποίο με τη σειρά του είναι μέλος του αντικειμένου **System**, μιας κλάσης που διευθετεί την έξοδο τιμών σε C#. Αυτός είναι ο λόγος που υπάρχουν στο παράδειγμά μας εντολές της μορφής:

```
System.Console.WriteLine ("κείμενο")
```

Το παράδειγμα περιλαμβάνει επίσης μια εντολή **if-else**, η οποία λειτουργεί όπως σε κάθε άλλη γλώσσα προγραμματισμού. Αν και οι εντολές C# θα αναλυθούν σε έκταση αργότερα, εδώ απλά σημειώνεται πως αν ισχύει η υπόθεση του if θα εκτελεστεί το σημειούμενο συμπέρασμα, διαφορετικά θα εκτελεστεί το συμπέρασμα που συνοδεύει το else. Αν λοιπόν ο Χρήστης στην γραμμή εκτέλεσης έδωσε σαν args[0] κάποια συμβολοσειρά που αρχίζει (αυστηρά) με το string “JAVA” τότε η συνάρτηση **StartsWith** θα επαληθευτεί και θα εκτελεσθεί το συμπέρασμα που αντίστοιχα σημειώνεται, διαφορετικά θα εκτελεσθεί το συμπέρασμα του else.

2.3 ΣΥΜΒΟΛΙΣΜΟΙ ΚΑΙ ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Τα προγράμματα C# συγκροτούνται από διάφορους **συμβολισμούς (Lexical Issues)** αλλά και **παρεχόμενες λέξεις κλειδιά** αυτής της ίδιας της γλώσσας **(Keywords),** όπως και στη Java. Μέσα στους βασικούς συμβολισμούς είναι και οι **Τελεστές (Operators),** οι οποίοι θα αναλυθούν αργότερα σε έκταση σε ειδικό κεφάλαιο του παρόντος.

2.3.1 ΟΡΙΣΜΟΣ ΜΕΤΑΒΛΗΤΩΝ

Βασικός συμβολισμός είναι τα **ονόματα (names, identifiers)** μεταβλητών σταθερών, κλάσεων, Μεθόδων και κάθε είδους παραμέτρων. Ένα όνομα αποτελείται από **αριθμούς** (0,1,2,…,9), **γράμματα** του λατινικού αλφαβήτου (μικρά και κεφαλαία), το χαρακτήρα **underscore** (_) ή και τον χαρακτήρα του **δολαρίου** ($). **Απαγορεύεται όμως να αρχίζουν με αριθμό**, ενώ υπάρχει **πλήρης διαχωρισμός** (**case-sensitive**) **μεταξύ μικρών και κεφαλαίων γραμμάτων** (π.χ. Some και some είναι δύο διαφορετικά ονόματα). Χαρακτηριστικά παραδείγματα ορθών ονομάτων είναι τα ακόλουθα:

```
SPEED       Speed       speed       $param      param$
  C4              a_long_name_variable          last_value
```

Λανθασμένα παραδείγματα ονομάτων είναι τα ακόλουθα, όπου ο Αναγνώστης πρέπει να προβληματιστεί για την αιτία του λάθους:

```
  3ij       iff(3)     jcp.some      class         if
cost-of-unit-in-$       Non/yes_variable        null
```

Δεν υπάρχει περιορισμός για το πλήθος των χαρακτήρων ενός ονόματος, αν και ένα περιορισμένο πλήθος αυτών είναι **σημαντικοί** για την C#. Φιλικά συμβουλεύεται ο Χρήστης να χρησιμοποιεί το πολύ μια-δυο δεκάδες χαρακτήρες για ονόματα. Ωστόσο, **ένα όνομα απαγορεύεται να ταυτίζεται με κάποια από τις λέξεις κλειδιά** (**keywords**) του μεταγλωττιστή της C#. Και μια λεπτομέρεια, απαγορεύεται επίσης ένα όνομα στην C# να έχει μια από τις τιμές **true**, **false** και **null**, διότι χρησιμοποιούνται σαν **συμβολικές σταθερές**…

Ειδικότερα, η δήλωση μεταβλητών σε ένα πρόγραμμα C#, είναι της μορφής

```
[ Modifier] DataType Identifier;
```

Ένα πολύ απλό παράδειγμα δήλωσης μεταβλητής σε C# είναι το ακόλουθο, με παρόμοια δομή με εκείνη της Java,

```
public int i;
```

Ενώ μπορούμε παράλληλα να κάνουμε και αρχικοποίηση, όπως φαίνεται παρακάτω:

```
int i=10;
```

2.3.2 ΤΥΠΟΙ ΜΕΤΑΒΛΗΤΩΝ

Η C# παρέχει μια πλούσια επιλογή από προκαθορισμένους τύπους τιμών οι οποίοι είναι συλλογικά γνωστοί σαν απλοί τύποι και περιλαμβάνουν τύπους ισοδύναμους με τους προγενέστερους διαθέσιμους της Java. Επιπλέον η C# παρέχει μη προσημασμένη (unsigned) έκδοση από όλους τους τύπους ακεραίων (integer) όπως και ένα νέο τύπο που ονομάζεται decimal. Το decimal είναι τύπος τιμών μεγάλη ακρίβεια σταθερού σημείου (high-precision fixed-point) σχεδιασμένο για τη χρήση υπολογισμών τους οποίους το θέμα της στρογγυλοποίησης σε συσχέτιση με την αριθμητική κινητού σημείου είναι προβληματική.

Κάθε ένας από του προκαθορισμένους τύπους τιμών είναι μια δομή υλοποιημένη μέσα στη .NET κλάση βιβλιοθήκης System. Για ευκολία, η C# παρέχει λέξεις κλειδιά (keywords) για να παραπέμπονται σε αυτούς τους απλούς τύπους. Και τα μεν keywords και τα δε ονόματα δομών μπορούν να χρησιμοποιηθούν μέσα σε ένα πρόγραμμα.

Στον πίνακα που ακολουθεί παρατίθενται οι προκαθορισμένοι τύποι λέξεων-κλειδιών και οι αντίστοιχες ισοδυναμίες τους με τη Java.

Πίνακας τύπων δεδομένων

Τύποι Java	C# keywords	Δομές .NET	Πεδίο Τιμών	Προεπιλεγμένη τιμή
String	string	System.String	συμβολοσειρά	" "
Boolean	Bool	System.Boolean	true ή false	false
Byte	Sbyte	System.Sbyte	8-bit προσημασμένος ακέραιος (-128 έως 127)	0
(N/A)	Byte	System.Byte	8-bit μη προσημασμένος ακέραιος (0 έως 255)	0
Short	Short	System.Int16	16-bit προσημασμένος ακέραιος (-32768 έως 32767)	0
(N/A)	ushort	System.Uint16	16-bit μη προσημασμένος ακέραιος (0 έως 65535)	0
Int	Int	System.Int32	32-bit προσημασμένος ακέραιος (-2147483648 έως 2147483647)	0
(N/A)	Uint	System.Uint32	32-bit μη προσημασμένος ακέραιος (0 έως 4294967295)	0
Long	Long	System.Int64	64-bit προσημασμένος ακέραιος (-9223372036854775808 έως 9223372036854775807)	0
(N/A)	ulong	System.UInt64	64-bit μη προσημασμένος ακέραιος (0 έως 18446744073709551615)	0

Float	Float	System.Single	32-bit διπλής ακρίβειας κινητού σημείου	0
Double	double	System.Double	64-bit διπλής ακρίβειας κινητού σημείου	0
(N/A)	decimal	System.Decimal	128-bit μεγάλη ακρίβεια δεκαδικός αριθμός με 28 καθοριστικά ψηφία	\u0000 0
Char	Char	System.Char	2-bit Unicode	0

2.3.2.1 ΣΤΑΘΕΡΕΣ

Οι σταθερές της C# μπορεί να έχουν τιμές ακέραιες, δεκαδικές, λογικές, χαρακτήρες ή και παραθέσεις χαρακτήρων. Μέσα στις τιμές των σταθερών (**constants** ή **literals**) μπορεί να εμπλέκονται και ειδικοί χαρακτήρες στο τέλος της τιμής, σαν πρόσφυμα (**suffix**) κατά κανόνα για να δηλωθεί το μήκος σε bits της σταθερής τιμής.

2.3.2.1.1 ΑΚΕΡΑΙΕΣ ΣΤΑΘΕΡΕΣ

Οι ακέραιες σταθερές της C# είναι οι γνωστοί μας ακέραιοι και απασχολούν μέγεθος μνήμης ανάλογα τον επί μέρους τύπο τους. Ωστόσο, ο Αναγνώστης πρέπει να λάβει υπόψη του πως μπορεί μια ακέραια σταθερά να έχει δηλωθεί με μήκος που ανταποκρίνεται σε 8-bits, δηλαδή να είναι τύπου byte ή και short σαν 16-bits, αλλά ο Διερμηνέας να προτιμά να διαθέτει μνήμη σαν 32-bits τιμές, απλά και μόνον διότι **αριστοποιεί** έτσι την πρώτη νόρμα των προγραμμάτων, δηλαδή τον αναμενόμενο υπολογιστικό χρόνο, εφόσον βέβαια η μνήμη επαρκεί και το μήκος λέξης της μηχανής (**word size**) είναι των 32-bits!

Προσοχή, οι σταθερές που είναι τύπου **long** θα πρέπει στο τέλος της τιμής να έχουν σαν **suffix** τον ειδικό χαρακτήρα **L** (ή **l**), διαφορετικά το σύστημα τις νομίζει κατά κανόνα int. Για παράδειγμα, αν μέσα σε πρόγραμμα JAVA σημειωθούν οι τιμές 150 και 150L, αν και αριθμητικά είναι ίσες μεταξύ τους, ωστόσο δεν ταυτίζονται από πλευράς έκτασης της μνήμης που απαιτούν. Η πρώτη μπορεί σε ένα σύστημα των 32-bits να απασχολήσει 2 ή και 4 bytes, ενώ η δεύτερη θα απασχολήσει αναγκαστικά 8 bytes κύριας μνήμης…

2.3.2.1.2 ΔΕΚΑΔΙΚΕΣ ΣΤΑΘΕΡΕΣ

Μια δεκαδική τιμή (ή σταθερά κινητής υποδιαστολής) περιλαμβάνει το πρόσημο (το + εννοείται), το ακέραιο μέρος και το δεκαδικό μετά την υποδιαστολή που παρίσταται με την τελεία (.). Ακολουθούν το πρότυπο IEEE-754 και είναι επομένως δεκτός και ο συμβολισμός σε επιστημονική εκθετική μορφή με τη βοήθεια του ειδικού χαρακτήρα e (ή και E). Επομένως στην C# είναι δεκτές οι ακόλουθες παραστάσεις δεκαδικών τιμών:

- Δεκαδική παράσταση (**Σtandard notation**): Με βάση τον τύπο **pn.m**, p: πρόσημο (το + εννοείται), n: ακέραιο μέρος και m: δεκαδικό μέρος
- Επιστημονική παράσταση (scientific notation): Με βάση τον τύπο **pn.mEsi = pn.m*10^{si}**, όπου p: πρόσημο της βάσης, n: ακέραιο μέρος βάσης, m: δεκαδικό μέρος βάσης, s: πρόσημο εκθέτη (το + εννοείται), i: ακέραιος εκθέτης και E (ή e): η βάση του εκθέτη ίση με δέκα (10).

Επιπρόσθετα των δύο μορφών παράστασης, η C# θεωρεί αυτόματα κάθε δεκαδική σταθερά σαν τιμή διπλής ακρίβειας (τύπου double) με απαιτήσεις μνήμης της τάξης των 64-bits, εκτός αν την τιμή ακολουθεί το suffix **F** (ή **f**), οπότε η τιμή θεωρείται απλή δεκαδική σταθερά των 32-bits, τύπου float.

Ακολουθούν χαρακτηριστικά παραδείγματα δεκαδικών σταθερών της C# και στις δύο δυνατές παραστάσεις:

`3.14159`	τύπου double (64-bits)
`3.141597`	τύπου float (32-bits)
`0.077e2`	τύπου double, ίση με $0.077*10^{2}$
`33E-15F`	τύπου float, ίση με $33*10^{-15}$
`3E+200`	τύπου double, ίση με $3*10^{200}$
`-31459E-05`	τύπου double, ίση με $-314159*10^{-5}$

Τέλος, πρέπει να σημειωθεί πως οι σταθερές τύπου float έχουν καλύτερους χρόνους εκτέλεσης από τις σταθερές τύπου double, αλλά χάνουν σε ακρίβεια ή σε μέγεθος τιμής.

2.3.2.1.3 ΣΤΑΘΕΡΕΣ ΧΑΡΑΚΤΗΡΩΝ

Ο Αναγνώστης που γνωρίζει Java, πρέπει να προσέξει ιδιαίτερα για τις σταθερές τιμές χαρακτήρων της C#, όπου εδώ μια τέτοια τιμή απασχολεί 16-bits μνήμη, με χαρακτήρες αποκλειστικά από το αλφάβητο **Unicode**. Βέβαια, οι γνωστοί μας χαρακτήρες ASCII ευτυχώς παραμένουν στις θέσεις από 0 μέχρι 127 και το αλφάβητο ISO-Latin-1 παραμένει από 0 μέχρι 255. Αλλά βέβαια, το Unicode έχει ενσωματώσει και δεκάδες άλλους χαρακτήρες από διάφορες φυσικές γλώσσες όπως ελληνικούς, Αραβικούς, Ιαπωνικούς, κλπ. με άμεση συνέπεια να μην είναι δυνατόν η αντιπροσώπευσή τους σε 8-bits και να φθάσουμε έτσι στα 16-bits!

Οι σταθερές τύπου χαρακτήρα ή δίνονται μεταξύ απλών εισαγωγικών (όπως οι χαρακτήρες ASCII) ή δίνονται με τη βοήθεια **ακολουθίας διαφυγής** (**escape sequence**) σε οκταδικό ή δεκαεξαδικό σύστημα. Για παράδειγμα ο χαρακτήρας «α» μπορεί και να δοθεί σαν οκταδικός με μορφή ακολουθίας διαφυγής '\141»' ή και στο δεκαεξαδικό σαν '\u0061'.

Ο Πίνακας με τις Ακολουθίες Διαφυγής της C# που παρατίθεται προσδιορίζει επίσης και άλλες τιμές όπως '\n', η οποία αν δοθεί σε έξοδο προγράμματος C#, θα προκαλέσει εμφάνιση δεδομένων σε νέα γραμμή (new line). Γενικά, μια ακολουθία διαφυγής αρχίζει με τον χαρακτήρα backslash (\) και παρόλο που περιέχει φαινομενικά περισσότερους από έναν χαρακτήρες (π.χ. η τιμή '\141' έχει 4 χαρακτήρες). Ωστόσο παριστά έναν ακριβώς χαρακτήρα μέσα στο αλφάβητο Unicode. Τέλος, η χρήση των σταθερών τύπου χαρακτήρα, μπορεί σε πολλές περιπτώσεις να δίνεται και σαν ακέραιος ή και να υπακούει σε τελεστές πρόσθεσης και αφαίρεσης.

Για παράδειγμα, μέσα σε πρόγραμμα C# είναι δυνατόν να δοθεί η ακέραια τιμή 88 σε μια μεταβλητή τύπου χαρακτήρα, το οποίο όμως ισοδυναμεί με τον χαρακτήρα 'X', ή και να δοθεί σαν τιμή στην μεταβλητή αυτή, η τιμή που έχει συν ένα, οπότε είναι ισοδύναμο με το να δοθεί η τιμή 'Y' που είναι και η επόμενη βέβαια της τιμής 'X' που υπήρχε…

Πίνακας Ακολουθιών Διαφυγής της C#	
ΑΚΟΛΟΥΘΙΑ ΔΙΑΦΥΓΗΣ	**ΕΡΜΗΝΕΙΑ/ΕΝΕΡΓΕΙΑ**
\n	**Τερματισμός γραμμής (LF)**
\r	**Χαρακτήρας Επιστροφής (CR)**
\0	**Κενός Χαρακτήρας**
\'	**Χαρακτήρας απλού εισαγωγικού (')**
\"	**Χαρακτήρας διπλού εισαγωγικού (")**
****	**Χαρακτήρας Backslash (\)**
\f	**form feed Χαρακτήρας**
\t	**Χαρακτήρας tab**
\b	**Χαρακτήρας backspace**
\a	**Προειδοποίηση (alert)**
\uyyyy	**Δεκαεξαδική διάταξη (yyyy) χαρακτήρα**

2.3.2.1.4 ΛΟΓΙΚΕΣ ΣΤΑΘΕΡΕΣ

Η C# διαθέτει ένα απλό είδος Λογικών Τιμών (**Bool**), τις τιμές **true** ή **false**. Οι τιμές αυτές είναι στην ουσία συμβολικές σταθερές και επιστρέφονται παντού στη C# όπου υπάρχει απάντηση σε κάποια υπόθεση, όπως για παράδειγμα στην υπόθεση της εντολής if. Μάλιστα, μέσα στα πλαίσια μιας δράσης του τύπου Write σε μια μεταβλητή του τύπου boolean, θα εμφανισθεί κανονικά στην έξοδο η τιμή true ή false. Αυτό σημαίνει απλά, πως οι τιμές αυτές δεν έχουν καμία σχέση με τις ακέραιες τιμές 1 ή 0, αντίστοιχα, όπως συμβαίνει σε άλλες γλώσσες προγραμματισμού.

2.3.2.1.5 ΠΑΡΑΘΕΣΕΙΣ ΧΑΡΑΚΤΗΡΩΝ (STRINGS)

Κάθε παράθεση χαρακτήρων του Unicode αλφαβήτου, που αρχίζει και τελειώνει με διπλά εισαγωγικά (") είναι στην C# σταθερά παράθεση χαρακτήρων (String), όπως και σε άλλες γλώσσες προγραμματισμού. Ωστόσο, ο μεταγλωττιστής της C# μετατρέπει αυτόματα σε **αντικείμενα** αντίστοιχα τα strings, ενώ είναι στη διάθεση του Προγραμματιστή αρκετές δυνατότητες που αφορούν τα ειδικά αυτά αντικείμενα (string manipulation). Ένα string, εκτός από τους κανονικούς χαρακτήρες του Unicode, μπορεί να περιέχει και ακολουθίες διαφυγής της C#, οπότε αν η παράθεση αυτή χαρακτήρων εμφανισθεί στην οθόνη με κάποια Μέθοδο Εξόδου, η ακολουθία διαφυγής θα προκαλέσει την αντίστοιχη λειτουργικότητα.

Ακολουθούν χαρακτηριστικά παραδείγματα από **string literals**, που σε ορισμένα ενυπάρχουν και **ακολουθίες διαφυγής**:

```
"Κόστος ανά μονάδα"
"Δώσε την τιμή του επιτοκίου : \n ==>"
```

```
"Το παρασύνθημα είναι \"7431\""
"Ένας ιαπωνικός χαρακτήρας: \ua432!"
"*********"
```

Τέλος, ο Αναγνώστης πρέπει να λάβει σοβαρά υπόψη του, πως τα strings της C# δεν διαθέτουν κάποιο τρόπο δήλωσης συνέχειας του string σε επόμενη γραμμή (line continuation). Αυτό σημαίνει πως **μέσα στα πηγαία προγράμματά μας το string πρέπει να αρχίζει και να τελειώνει στην ίδια γραμμή**…

2.3.2.1.6 ΤΕΛΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ

Η έννοια του **final** υπάρχει στην C# και σε επίπεδο μεταβλητής, αλλά και σε επίπεδο Μεθόδου και Κλάσης με τις **ReadOnly** και **Sealed** αντίστοιχα. Αργότερα, μέσα στα πλαίσια της **Κληρονομικότητας (Inheritance)**, θα γίνει ανάλυση του Sealed για Μεθόδους και Κλάσεις.

2.3.2.1.7. ΜΗΤΡΕΣ

Μια μήτρα (**array**) της C# είναι ένας ειδικός τύπος σύνθετης μεταβλητής, η οποία κάτω από ένα κοινό όνομα, φιλοξενεί πολλές άλλες **όμοιες μεταβλητές**. Η μήτρα μπορεί να έχει μία, δύο ή και περισσότερες διαστάσεις, οι οποίες σημειώνονται μεταξύ των συμβόλων **[** και **]**. Οι διαστάσεις συμβολίζονται με ακέραιες μεταβλητές (δείκτες μητρών) ή και με ακέραιες μη αρνητικές τιμές. Στην C#, οι δείκτες (**subscripts**) αρχίζουν για κάθε διάσταση από την τιμή μηδέν μέχρι και την μέγιστη τιμή της διάστασης πλην ένα.

Η δήλωση μιας μονοδιάστατης ή και πολυδιάστατης μήτρας γίνεται σε δύο στάδια. Στο πρώτο στάδιο ορίζεται το είδος των τιμών που φιλοξενεί η μήτρα, ενώ στο δεύτερο το πλήθος των διαστάσεων και η μέγιστη τιμή κάθε διάστασης. Ο γενικός συντακτικός τύπος δήλωσης μήτρας είναι ο ακόλουθος:

Τύπος_Τιμών_Μήτρας Όνομα_Μήτρας [ ] [ ]…[ ];
Όνομα_Μήτρας=new Τύπος_Τιμών_Μήτρας [max1] [max2]…[maxk];

ή ισοδύναμα, σε μια εντολή δήλωσης που συνδυάζει και τα δύο στάδια:

Τύπος_Τιμών_Μήτρας Όνομα_Μήτρας [ ] [ ]…[ ]=
new Τύπος_Τιμών_Μήτρας [max1] [max2] … [maxk];

ή και ισοδύναμα, τα σύμβολα **[** και **]**, να σημειώνονται πριν από το όνομα της μήτρας (το οποίο δεν συμβουλεύουμε):

Τύπος_Τιμών_Μήτρας [] []…[] Όνομα_Μήτρας=
new Τύπος_Τιμών_Μήτρας [max1] [max2]…[maxk];

Εύκολα παρατηρεί ο Αναγνώστης, την χρήση του τελεστή **new**, ο οποίος δεσμεύει απλά κύρια μνήμη για τα διαμερίσματα τιμών της Μήτρας, σε συνεχή έκταση και τόση, όση το πλήθος των μεταβλητών που φιλοξενεί η Μήτρα, επί τα bytes που αντιστοιχούν στον Τύπο τιμών της Μήτρας:

ram(Όνομα_Μήτρας) = ram (Τύπος_Τιμών_Μήτρας)*max1*…*maxk

όπου **maxi** το πλήθος των στοιχείων της διάστασης i και **ram(x)** συνάρτηση ίση με τα bytes σε κύρια μνήμη που απαιτούνται για την τοποθέτηση της τιμής του x. Οπωσδήποτε, ο ειδικός τελεστής **new** θα αναλυθεί αργότερα, μαζί με τους άλλους τελεστές της C#, ωστόσο, ο Αναγνώστης αρκεί να γνωρίζει προς το παρόν πως ο τελεστής αυτός χρησιμοποιείται για δημιουργία αντικειμένων και δέσμευση κύριας μνήμης γι' αυτά.

Ακολουθούν απλά παραδείγματα δηλώσεων μητρών διαφόρων τύπων δεδομένων με διάφορους τρόπους:

```
int products[];
products = new int[1000];
float profits[] = new float[1000];
double costs[][] = new double[100][50];
short[][][] other = new short[10][20][30];
```

όπου η μήτρα products είναι μονοδιάστατη με 1000 στοιχεία τύπου int και ορίζεται σε δύο στάδια. Η μήτρα profits είναι μονοδιάστατη 1000 στοιχείων με τιμές τύπου float και ορίζεται με συνθετικό τύπο σε μια εντολή. Η μήτρα costs είναι τιμών double, δισδιάστατη 100x50, ενώ η μήτρα other είναι τύπου short οι τιμές των μεταβλητών της, που είναι οργανωμένες σε τρεις διαστάσεις 10x20x30.

Το τυχαίο στοιχείο μιας μήτρας μέσα στα προγράμματα C#, συμβολίζεται απλά με το όνομα της μήτρας και αντίστοιχες τιμές των δεικτών των διαστάσεων:

```
Ονομα_Μήτρας [i1] [i2]...[ik];
```

Για παράδειγμα, τα ακόλουθα στοιχεία έχουν νόημα σε ένα πρόγραμμα C# με βάση τις δηλώσεις τους:

```
products[999],profits[0],costs[0][49],other[1][2][3]
```

ενώ τα ακόλουθα στοιχεία μητρών προκαλούν **προγραμματιστικά λάθη**:

```
products[1000],profits[-3],costs[101][49],other[0][0][31]
```

Η μήτρα εφόσον χρησιμοποιείται σαν αντικείμενο, με τη βοήθεια του τελεστή new, αυτόματα θέτει στην κύρια μνήμη όλες οι τιμές των μεταβλητών που φιλοξενεί, ίσες με το ουδέτερο στοιχείο της άλγεβρας του τύπου τιμών της μήτρας. Για παράδειγμα, η δήλωση:

```
double matrix[]=new double[10];
```

όχι μόνον θα προσδιορίσει κύρια μνήμη σε συνεχή έκταση για τα στοιχεία της μήτρας matrix με τη σειρά:

```
matrix[0], matrix[1],..., matrix[9]
```

αλλά θα προκαλέσει και αυτόματη αρχικοποίηση τιμών, θέτοντας σε όλες την τιμή 0.0, δηλαδή το δεκαδικό μηδέν σε 64-bits μήκος. Ωστόσο, υπάρχει δυνατότητα στην C#, όπως και στην Java (όχι όμως με τον ίδιο τρόπο), να δίνει κανείς αρχικές τιμές σε μήτρα κάνοντας χρήση του αρχικοποιητή τιμών μήτρας

(**array initializer**) και μάλιστα **χωρίς τον τελεστή new**, με βάση τον συντακτικό τύπο (για μια και δύο διαστάσεις, για περισσότερες διαστάσεις είναι άχρηστος λόγω συντακτικής πολυπλοκότητας):

```
Τύπος_Τιμών_Μήτρας Όνομα_Μήτρας [ ] =
    { r0, r1, …, rmax1-1};  // Μονοδιάστατη
Τύπος_Τιμών_Μήτρας Όνομα_Μήτρας [ ] [ ] =
    { { ρ0^0, ρ1^0, …, ρmax2-1^0 }
  { ρ0^1, ρ1^1, …, ρmax2-1^1 },
      ……
  { ρ0^max1-1, ρ1^max1-1, …, ρmax2-1^max1-1 } }      // Δυσδιάστατη
```

όπου max1, max2 το πλήθος των στοιχείων της πρώτης (γραμμές) και της δεύτερης (στήλες) διάστασης. Η αρχική τιμή r_i αναφέρεται στο στοιχείο [i] της μονοδιάστατης μήτρας, ενώ η αρχική τιμή ρ_j^i αντιστοιχεί στο στοιχείο [i][j] της δυσδιάστατης μήτρας. Ακολουθούν παραδείγματα δηλώσεων μητρών με αρχικές τιμές;

```
int months[]={31,28,31,30,31,30,31,31,30,31,30,31};
boolean yes_no[] = {false, true, true, false};
double values[] = {1.2*4.5, 2.3, 1.3*4.2,-1.0*4.0};
int twodim[][] = {{1,2,3},{5,4,6},{7,8,9},{0,1,0}};
```

όπου η μήτρα months (προφανώς) περιγράφει τις μέρες που έχει κάθε μήνας, η λογική μήτρα yes_no έχει 4 στοιχεία, η μήτρα values είναι μονοδιάστατη με 4 στοιχεία τύπου double, όπου οι αρχικές τιμές σημειώνονται σαν αποτέλεσμα κάποιων πράξεων (παραστάσεων), ενώ η ακέραια μήτρα twodim έχει διαστάσεις 4x3 και ανάπτυγμα:

```
1  2  3          [0][0]  [0][1]  [0][2]
5  4  6          [1][0]  [1][1]  [1][2]
7  8  9          [2][0]  [2][1]  [2][2]
0  1  0          [3][0]  [3][1]  [3][2]
```

Στη C#, όπως και στην Java, η καταχώριση μέσα στη μνήμη πολυδιάστατων μητρών γίνεται **κατά γραμμή** και όχι κατά στήλη όπως σε άλλες γλώσσες προγραμματισμού. Για παράδειγμα, η καταχώριση τιμών των στοιχείων της μήτρας twodim ακολουθεί τη σειρά:

1 2 3 5 4 6 7 8 9 0 1 0

σε συνεχή έκταση μνήμης μήκους (12*32) bits.

Η C# στην ουσία θεωρεί τις πολυδιάστατες μήτρες σαν μήτρες μητρών. Για παράδειγμα τη δισδιάστατη μήτρα ακεραίων τιμών, την θεωρεί σαν μια μήτρα μονοδιάστατη που το κάθε στοιχείο της είναι μια μονοδιάστατη μήτρα ακεραίων τιμών. Στην περίπτωση αυτή, **μπορεί ο Χρήστης να σημειώσει την**

δεύτερη διάσταση αναλυτικά. Για παράδειγμα, οι ακόλουθες δηλώσεις της δισδιάστατης μήτρας matrixd είναι τελείων ισοδύναμες:

```
int matrixd[][] = new int[5][6];
// Αναλυτική δήλωση:
int matrixd[][] = new int[5][];
matrixdp[0] = new int[6];
matrixdp[1] = new int[6];
matrixdp[2] = new int[6];
matrixdp[3] = new int[6];
matrixdp[4] = new int[6];
```

Στο σημείο αυτό, ο Αναγνώστης δίκαια θα αναρωτηθεί ποια είναι η χρησιμότητα μιας τέτοιας αναλυτικής δήλωσης. Η απάντηση είναι πως η C# έχει τη δυνατότητα δήλωσης μη κανονικών μητρών (irregular arrays), όπου δεν είναι το ίδιο πλήθος στηλών για κάθε γραμμή. Για παράδειγμα η ακόλουθη δήλωση:

```
int matrixd[][] = new int[5][];
matrixdp[0] = new int[6];
matrixdp[1] = new int[5];
matrixdp[2] = new int[4];
matrixdp[3] = new int[3];
matrixdp[4] = new int[7];
```

προσδιορίζει μια μήτρα μη κανονική, όπου το πλήθος των στηλών δεν είναι το ίδιο για κάθε γραμμή (π.χ., η τελευταία γραμμή έχει 7 στοιχεία, ενώ η πρώτη 6).

Η έννοια των μη κανονικών μητρών σε ορισμένες περιπτώσεις, **είναι κριτήριο αποφασιστικής μείωσης της κύριας μνήμης που απαιτεί το πρόγραμμα** για την εκτέλεσή του, το οποίο είναι ένας από τους στόχους της αριστοποίησης των προγραμμάτων.

2.3.2.1.8 ΜΕΤΑΒΛΗΤΕΣ ΤΥΠΟΥ STRING

Μια σύνθετη μεταβλητή της C# είναι επίσης εκείνη που παίρνει τιμές συμβολοσειρές (strings), όπως ήδη έχει αναλυθεί σε προηγούμενες παραγράφους. Μια τέτοια μεταβλητή μπορεί επίσης να είναι μήτρα από strings. Σε κάθε περίπτωση υπάρχουν πολλές ευκολίες διαχείρισης συμβολοσειρών από την παρεχόμενη κλάση **string** της C#, η οποία μάλιστα είναι **τελική**, χωρίς δηλαδή υποκλάσεις. Στο παρόν σύγγραμμα, σε επόμενο ειδικό κεφάλαιο, θα γίνει εκτενής ανάλυση των δυνατοτήτων της ειδικής αυτής κλάσης. Εδώ, προς τα παρόν, θα δοθούν οι πρώτες απλές έννοιες των μεταβλητών τύπου string, όσες είναι αναγκαίες για την κανονική παρακολούθηση της ύλης από τον Αναγνώστη.

Ήδη έχει αναφερθεί πως οι σταθερές (literals) παραθέσεις χαρακτήρων είναι μέσα από το Unicode και πώς στην ουσία είναι **αντικείμενα**. Η C# δίνει δύο κλάσεις για την χρήση συμβολοσειρών:

- Η κλάση **string**, όπου σε κάθε αλλαγή τιμής ή του μήκους της, μιας τέτοιας μεταβλητής, στην ουσία ορίζεται αυτόματα άλλο αντικείμενο και η τιμή στην κύρια μνήμη αλλάζει βέβαια και αυτή μέρος. Η δήλωση μιας τέτοιας μεταβλητής είναι απλή:

```
string Ονομα_Μεταβλητής= "τιμή string";
```

Για παράδειγμα, οι ακόλουθες εντολές, δίνουν την ίδια ακριβώς έξοδο:

```
string example = "Λατρεύω τις μηχανές";
System.Console.WriteLine(example);
System.Console.WriteLine ("Λατρεύω τις μηχανές");
```

Αργότερα, θα γνωρίσουμε πολλές Μεθόδους των κλάσεων string, όπως για παράδειγμα την μέθοδο **length** που δίνει το μήκος της τιμής μιας μεταβλητής τύπου string.

2.3.2.1.9 Η ΔΟΜΗ STRUCT

Ένας τύπος struct, μπορεί να δηλώσει συναρτήσεις δημιουργίας constructor, σταθερές, πεδία, ιδιότητες, δείκτες, τελεστές και μεθόδους. Αν και τα χαρακτηριστικά του struct, μοιάζουν με αυτά της κλάσης, παρ' όλα αυτά η διαφορά τους φαίνεται από τα εξής τρία συμπεράσματα.

- Μία struct κατανέμει μνήμη πάνω στη στοίβα (stack) Εάν αυτό είναι μέλος ή σωρός (heap) εγκαταστημένων αντικειμένων τότε είναι μια κλάση.
- Η μνήμη διατίθεται σε μια struct στη περίπτωση που περικλείει τα δεδομένα μελών, όχι αναφορές για τα δεδομένα.
- Τα structs είναι διευθετημένα να διαγράφονται εφόσον χάσουν το σκοπό τους, δεν παραμένουν ως απορρίμματα

Στο παρακάτω παράδειγμα παρουσιάζεται τμήμα κώδικα, όπου δηλώνεται η δομή struct.

```
public struct FileInfo
{ public long id;
  public string Name;
  public string LastName;
  public decimal Account;
}
 FileInfo File1;
 File1= new FileInfo();
 File1.id=10;
 File1.Name="John";
 File1.LastName="Newton";
 File1.Account=50;
```

αρχικοποιείται δίνοντας τιμές στις μεταβλητές, ενώ παράλληλα, δεσμεύεται και μνήμη όπου εκεί αποθηκεύονται οι τιμές.

2.3.2.1.10 Η ΔΟΜΗ ENUMERATION

Η δομή enumeration, ορίζει ένα διακριτό τύπο που αποτελείται από ένα σύνολο από ονομαστικές σταθερές. Ένα απλό παράδειγμα ορισμού του enumeration είναι το παρακάτω

```
enum month {January, February, March};
```

Αυτό που πρέπει να επισημάνουμε είναι ότι τα στοιχεία του enumeration είναι τύπου int και το πρώτο στοιχείο έχει την τιμή μηδέν και το επόμενο αυξάνεται κατά ένα. Έτσι το παραπάνω παράδειγμα γίνεται ως εξής:

```
enum month {January=1, February, March};
```

Μπορούμε επίσης, να προσθέσουμε διπλές τιμές όπως φαίνεται παρακάτω.

```
enum month {January=31, February=28, March=31};
```

2.3.3 ΟΡΑΤΟΤΗΤΑ ΜΕΤΑΒΛΗΤΩΝ

Όπως ήδη έχει αναφερθεί, η C# έχει ελάχιστες εντολές και βασίζεται κατά κανόνα σε **συναρτήσεις/υποπρογράμματα** (functions/subroutines) που υπάρχουν σε βιβλιοθήκες που συνοδεύουν τον μεταγλωττιστή. Βέβαια, στον Αντικειμενοστρεφή Προγραμματισμό, οι συναρτήσεις είναι ενταγμένες σε προγραμματιστικές ενότητες, όπως είναι οι κλάσεις, οπότε αντί να ονομάζονται συναρτήσεις, καλούνται **Μέθοδοι**. Ακόμη και η βασική συνάρτηση **Main()** θεωρείται Μέθοδος και επομένως θα πρέπει και αυτή να ανήκει σε κάποια κλάση! **Ιδιαίτερη προσοχή να έχουν όλες οι μέθοδοι το πρώτο τους γράμμα κεφαλαίο.**

Οι μεταβλητές C# δηλώνονται μέσα στα πλαίσια μιας προγραμματιστικής ενότητας (**block**), το οποίο αρχίζει με το **curly brace** ({) και κλείνει πάντα με το κλείσιμο του ιδίου χαρακτήρα (}). Ακριβώς ένα block προσδιορίζει και τη **θέα** των μεταβλητών του, όπου κάθε φορά που η λειτουργικότητα του προγράμματος αρχίζει ένα block, ορίζεται και η αντίστοιχη θέα (scope) των μεταβλητών που υπάρχουν. Υπάρχουν **δύο μεγάλες κατηγορίες θέας** στην C#, εκείνη που ορίζεται από την **Κλάση** και η περιορισμένη που προσδιορίζεται από την υπάρχουσα **Μέθοδο**. Στο σημείο αυτό, για να γίνουν κατανοητότερες οι έννοιες αυτές, ορίζονται στη συνέχεια η έννοια του **Κύκλου Ζωής** (**Lifetime**) και **Ορατότητας** (**Visibility**) μιας μεταβλητής:

- **Κύκλος Ζωής** μιας μεταβλητής είναι η διάρκεια διατήρησης του χώρου κύριας μνήμης αποθήκευσης της τιμής της. Η διάρκεια αυτή εξαρτάται από την έννοια της **Τοπικής** (**Local**) και **Σφαιρικής** (**Global**) μεταβλητής. Τοπική μεταβλητή είναι εκείνη που η τιμή της καταλαμβάνει νέο πεδίο κύριας μνήμης κάθε φορά που η λειτουργικότητα εισέρχεται στο block που ορίζεται η παρούσα μεταβλητή, ενώ όταν η λειτουργικότητα εξέρχεται του block, η μεταβλητή χάνει το πεδίο ορισμού της που είχε στη μνήμη. Αντίθετα, η Σφαιρική μεταβλητή διατηρεί δικό της σταθερό πεδίο μνήμης,

φαινόμενο που μπορεί να επεκταθεί και σε όλη την επικράτεια του προγράμματος.

- **Ορατότητα** μιας μεταβλητής είναι η ικανότητα αναγνώρισης της τιμής της από block σε block. Όπως θα δούμε, υπάρχει δυνατότητα ορατότητας σε διάφορες συνδυαστικές χρησιμοποιώντας κατάλληλες δηλώσεις. Ωστόσο, μέσα στα πλαίσια μιας Μεθόδου, κάθε μεταβλητή είναι ορατή στο block που ανήκει και σε κάθε εσωτερικό block, ενώ παύει να είναι ορατή σε κάθε εξωτερικό block.

Είναι λοιπόν φανερό πως τα blocks δημιουργούν διαπλεκόμενες δομές και σχηματισμούς (**nesting** φαινόμενο). Για παράδειγμα, στο ακόλουθο τμήμα προγράμματος:

```
{
int x;
x=3;
if (x==3)
   {
     int z=4;
     System.Console.WriteLine("x=" + x + "z=" + z);
   }
x=x*z; //Λάθος ορατότητας του z.
System.Console.WriteLine("x="+x);
}
```

επιχειρείται σημείωση της μεταβλητής Z σε εξωτερικό block, όπου βέβαια δεν έχει ορατότητα και στην ουσία είναι μια άγνωστη μεταβλητή για τον αντίστοιχο κώδικα… Αντίθετα, η μεταβλητή X είναι σφαιρική και επομένως ορατή και στο δικό της block και σε κάθε εσωτερικό.

Στην C# **μια μεταβλητή μπορεί να χρησιμοποιηθεί μόνον αφού έχει προηγηθεί η δήλωσή της**. Η δήλωση βέβαια μπορεί να γίνει οπουδήποτε, αν και καλό είναι (για λόγους δομημένου προγραμματισμού) να δηλώνεται στην αρχή της προγραμματιστικής ενότητας που ανήκει (declaration part). Για παράδειγμα, στο ακόλουθο τμήμα προγράμματος:

```
{
  int x,y=3;
  x=y*z;      //Λάθος ορατότητας του z.
  int z=1;
}
```

επιχειρείται χρήση της μεταβλητής z πριν αυτή δηλωθεί, το οποίο βέβαια είναι λάθος.

Τέλος, ένα σημείο που πρέπει ο Αναγνώστης να προσέξει, ιδιαίτερα αν είναι γνώστης της JAVA, είναι πως στη C# **απαγορεύεται η δήλωση μεταβλητών με το ίδιο όνομα** ακόμα και όταν ανήκουν σε διαφορετικά blocks! Για παράδειγμα στο ακόλουθο τμήμα προγράμματος:

```
{  int x=1;
   {int x=2;  //Λάθος, υπάρχει συνωνυμία.
   }
}
```

επιχειρείται η δήλωση της μεταβλητής x και σε κάποιο εσωτερικό block, ενέργεια ορθή σε C++, λανθασμένη όμως σε JAVA, C#. παρόλο που το εξωτερικό x μπορεί να χαρακτηριστεί σφαιρικό, ενώ το εσωτερικό x σαν τοπικό…

2.3.4 ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Παρακάτω παρατίθεται ο πίνακας με τις δεσμευμένες λέξεις στην C#.

Abstract	Do	implicit	params	switch
As	Double	in	private	this
Base	else	int	protected	through
Bool	enum	interface	public	true
break	event	internal	readonly	try
byte	Explicit	is	ref	typeof
case	Extern	lock	return	uint
catch	false	long	sbyte	ulong
char	finally	namespace	seald	unchecked
Checked	fixed	new	Sort	unsafe
class	float	null	sizeof	usort
const	for	object	stackalloc	using
Continue	foreach	operator	static	virtual
Decimal	goto	out	string	void
Default	If	override	struct	while
Delegate				

2.3.5 ΔΙΑΧΩΡΙΣΤΕΣ (SEPARATORS)

Η C# χρησιμοποιεί ειδικούς χαρακτήρες σαν **διαχωριστές (Separators).** Ήδη έχουμε γνωρίσει τον χαρακτήρα (**;**) (**semicolon**), που διαχωρίζει εντολές C# μεταξύ τους. Σε ειδικό Πίνακα δίνονται οι ειδικοί αυτοί χαρακτήρες, καθώς και η σημασία που μπορούν να πάρουν σε διάφορες περιπτώσεις.

Πίνακας Διαχωριστών της C#		
ΣΥΜΒΟΛΟ	**ΟΝΟΜΑ**	**ΧΡΗΣΗ**
;	Semicolon	Τερματίζει μια εντολή.
.	Period	Διαχωρίζει ένα μέλος ενός προγραμ-ματιστικού ολονίου από το όλον.
,	Comma	Διαχωρίζει παραμέτρους μιας μεταβλητής και ενώνει σειριακά

		εντολές μεταξύ τους.
( )	Parentheses	Περικλείουν παραμέτρους μεθόδων, επίσης σαν παρενθέσεις πράξεων, αλλά και σε εντολές ελέγχου.
[ ]	Brackets	Για αποκλειστική χρήση στοιχείων μητρών.
{ }	Braces	Για να περικλείουν μια λογισμική παράγραφο ή και για δήλωση αρχικών τιμών μητρών.

2.4 ΑΣΚΗΣΕΙΣ

1. Ποιες είναι οι βασικές ομοιότητες στην μεταγλώττιση προγραμμάτων Java και C#;

2. Επισημάνετε τις διαφορές στον τρόπο μεταγλώττισης των προγραμμάτων Java και C#.

3. Μελετήστε τους παρακάτω κώδικες, και βρείτε πιθανά λάθη. Που οφείλονται αυτά;

Α. ΚΩΔΙΚΑΣ 1

```
using System;
namespace Con2
{ public class Class1
  { int j=14;
    static void Main(string[] args)
     { int i=12;
       int j=15
       for( int i=1;i<10;i++)
        {
           Console.WriteLine ("Value of i="+i);
           Console.WriteLine ("Value of j="+j);
           j=j+1;
           Console.ReadLine();
       }
           Console.WriteLine ("Value of i="+i);
           Console.ReadLine();
   }
 }
}
```

Β.ΚΩΔΙΚΑΣ 2

```
using System;
namespace Con2
{
  public class Class1
   {
```

```
        static void Main(string[] args)
        {  int i=12;
           int j=15
           int [] int=new int [-30];
           string []agg=new string[(50)];
            for( int i=1;i<10;i++)
            {
              Console.WriteLine ("Value of i="+i);
              Console.WriteLine ("Value of j="+j);j=j+1;
              agg[i]=j;
              Console.ReadLine();
            }
            Console.WriteLine ("Value of i="+i);
            Console.ReadLine();
        }
    }
}
```

ΚΕΦΑΛΑΙΟ 3ο

3. ΤΕΛΕΣΤΕΣ

3.1 ΕΙΣΑΓΩΓΗ

Οπωσδήποτε η C++ διαθέτει την μεγαλύτερη συλλογή τελεστών (**operators**) από κάθε άλλη γλώσσα προγραμματισμού, παλαιομοδίτικου ή σύγχρονου, σε σημείο «φλυαρίας» όπως έχει κατηγορηθεί από μηχανικούς λογισμικού. Η Java σαν άξια θυγατέρα της αλλά και η C# στην συνέχεια προσπάθησαν να απαλλαγούν από ορισμένες πολυτέλειες τελεστών της C++, αλλά δεν κατόρθωσαν να συμπεριλάβουν ένα-δυο ελλείψεις τελεστών, όπως για παράδειγμα τον τελεστή α^x (**ύψωσης σε δύναμη**), που υπάρχει σε κάθε γλώσσα προγραμματισμού πλην των Pascal, C++, JAVAκαι C#! Ευτυχώς όμως κατήργησαν ορισμένους τελεστές που ήταν ήδη σε αχρηστία στα προγράμματα της C++. Ο Αναγνώστης που γνωρίζει JAVA ναι μεν θα καταλάβει αμέσως τους συντακτικούς τύπους των τελεστών της C# θα πρέπει όμως να είναι ιδιαίτερα προσεκτικός γιατί υπάρχουν σημαντικές διαφορές των τελεστών των δύο αυτών γλωσσών προγραμματισμού.

Ανάλογα με το πλήθος των όρων (operators) που εμπλέκονται με τον τελεστή μπορούμε να έχουμε την κλασική κατηγοριοποίηση των τελεστών:

Μονήρεις (**unary**): Ονομάζονται και απλά μοναδιαίοι τελεστές και συντάσσονται με έναν όρο. Αν **r1**, **r2** μοναδιαίοι τελεστές που εφαρμόζονται στον όρο **x**, ισχύει η ανάγνωση από δεξιά προς τα αριστερά, δηλαδή ισχύει η σχέση:

r1r2x = r1(r2x)

Δυαδικοί (**binary**): Είναι τελεστές που χρησιμοποιούν δύο όρους, τον δεξιό και αριστερό όρο της έκφρασης. Εδώ ανήκει ο κύριος όγκος των τελεστών της JAVA και C#.

Τριαδικοί (**ternary**): Υπάρχει μόνον ο **Υπό συνθήκη τελεστής** (**conditional operator**), ο οποίος χρησιμοποιεί τρεις όρους.

Ωστόσο, η προηγούμενη κατηγοριοποίηση δεν βοηθά τον Αναγνώστη. Είναι καλύτερα να θεωρηθεί η ακόλουθη κατηγοριοποίηση, η οποία συνίσταται για λόγους καθαρά εκπαιδευτικούς:

- **Αριθμητικοί τελεστές (arithmetic operators)**
- **Ψηφιακοί τελεστές με bits (bitwise operators)**
- **Τελεστές σύγκρισης (relational operators)**
- **Λογικοί τελεστές (logical operators)**
- **Ειδικοί τελεστές (specific operators**)

Οι αριθμητικοί τελεστές διευθετούν τις βασικές πράξεις αριθμητικών τιμών και δεν εφαρμόζονται σε τύπους Boolean. Οι ψηφιακοί εκτελούν πράξεις σε τιμές ψηφιακές με μηδέν και ένα. Οι τελεστές σύγκρισης επιστρέφουν true ή false σαν αποτέλεσμα μιας σύγκρισης μεταξύ δύο τιμών. Οι λογικοί τελεστές

εκτελούν λογικές πράξεις μεταξύ εκφράσεων που δίνουν λογική τιμή false ή true. Τέλος, οι ειδικοί τελεστές στην JAVA και C# προσφέρουν πεδία μνήμης για τιμές σε αντικείμενα ή μετατρέπουν τιμές σε άλλο τύπο ή εκφράζουν ένα μέλος κάποιου οργανωμένου λογισμικού ολόνιου (π.χ., αντικειμένου) ή τέλος συμπερασματολογούν αν κάποιο αντικείμενο ανήκει σε κάποια υποκλάση. Τέλος, μέσα στα πλαίσια των τελεστών, συμπεριλαμβάνονται και τα ειδικά σύμβολα **( )** και **[ ]** που ανήκουν στους ειδικούς τελεστές με μέγιστη προτεραιότητα.

Στη συνέχεια δίνεται ο ειδικός Πίνακας Τελεστών της JAVA, με τους αντίστοιχους τελεστές C# ο οποίος συγκεντρώνει περιληπτικά όλους τους τελεστές που χρησιμοποιεί, με τις αντίστοιχες προτεραιότητες.

ΠΙΝΑΚΑΣ ΤΕΛΕΣΤΩΝ ΤΗΣ JAVA/C#

Τελεστής Java	**Τελεστής C#**	**Περιγραφή**
μέθοδος(λίστα παραμέτρων)	Μέθοδος (λίστα παραμέτρων)	Λίστα παραμέτρων συνάρτησης (μεθόδου)
κατασκευαστική(τιμές)	κατασκευαστική(τιμές)	Αρχικές τιμές ορισμού αντικειμένου
μήτρα[δείκτες]	μήτρα[δείκτες]	Στοιχείο μήτρας
αντικείμενο.μέλος	αντικείμενο.μέλος	Επιλογή μέλους
μεταβλητή++	μεταβλητή++	Επιθεματική αύξηση κατά 1
μεταβλητή--	μεταβλητή--	Επιθεματική μείωση κατά 1
++μεταβλητή	++μεταβλητή	Προθεματική αύξηση κατά 1
--μεταβλητή	--μεταβλητή	Προθεματική μείωση κατά 1
+παράσταση	+παράσταση	Πρόσημο (μονομελές συν) (παραλείπεται)
-παράσταση	-παράσταση	Πρόσημο (μονομελές πλην)
~παράσταση	~παράσταση	Δυαδικό συμπλήρωμα και λογική άρνηση
!παράσταση	!παράσταση	Λογικό συμπλήρωμα
New κλάση()	new κλάση()	Δέσμευση μνήμης για αντικείμενο κλάσης
New τύπος[]	new τύπος[]	Δέσμευση μνήμης για μήτρα ορισμένου τύπου
(type)παράσταση	(type)παράσταση	Ρητή μετατροπή τύπου
παράσταση*παράσταση	παράσταση*παράσταση	Πολλαπλασιασμός
παράσταση/παράσταση	παράσταση/παράσταση	Διαίρεση
παράσταση%παράσταση	παράσταση%παράσταση	Υπόλοιπο διαίρεσης
παράσταση+παράσταση	παράσταση+παράσταση	Πρόσθεση

string+string	string+string	Παράθεση συμβολοσειρών
παράσταση-παράσταση	παράσταση-παράσταση	Αφαίρεση
παράσταση>>x	παράσταση>>x	Δεξιά ολίσθηση των bits σε απόσταση x
παράσταση<<x	παράσταση<<x	Αριστερή ολίσθηση των bits σε απόσταση x
παράσταση>>>x	(N/A)	Δεξιά ολίσθηση αλλά χωρίς επέκταση προσήμου
παράσταση>παράσταση	παράσταση>παράσταση	Μεγαλύτερο (σύγκριση)
παράσταση>=παράσταση	παράσταση>=παράσταση	Μεγαλύτερο ή ίσο (σύγκριση)
παράσταση<παράσταση	παράσταση<παράσταση	Μικρότερο (σύγκριση)
παράσταση<=παράσταση	παράσταση<=παράσταση	Μικρότερο από (σύγκριση)
αντικείμενο instanceof C	αντικείμενο is typeof C	Αν το αντικείμενο ανήκει στην κλάση C
παράσταση==παράσταση	παράσταση==παράσταση	Αν οι δύο τιμές ταυτίζονται
παράσταση!=παράσταση	παράσταση!=παράσταση	Αν οι δύο τιμές είναι διάφορες μεταξύ τους
παράσταση&παράσταση	παράσταση&παράσταση	Ψηφιακό AND σε bits
παράσταση^παράσταση	παράσταση^παράσταση	Ψηφιακό αποκλειστικό OR σε bits
παράσταση|παράσταση	παράσταση|παράσταση	Ψηφιακό εγκλειστικό OR σε bits
παράσταση&&παράσταση	παράσταση&&παράσταση	Λογικό AND
παράσταση||παράσταση	παράσταση||παράσταση	Λογικό εγκλειστικό OR
παράσταση? παράσταση:παράσταση	παράσταση? παράσταση:παράσταση	Υπό συνθήκη τελεστής (εναλλα-κτική αποτίμηση υπό συνθήκη)
μεταβλητή=παράσταση	μεταβλητή=παράσταση	Εκχώρηση τιμής (ισότητα αντικατάστασης)
μεταβλητή+=παράσταση	μεταβλητή+=παράσταση	μεταβλητή=μεταβλητή+παράσταση
μεταβλητή-=παράσταση	μεταβλητή-=παράσταση	μεταβλητή=μεταβλητή-παράσταση
μεταβλητή*=παράσταση	μεταβλητή*=παράσταση	μεταβλητή=μεταβλητή*παράσταση
μεταβλητή/=παράσταση	μεταβλητή/=παράσταση	μεταβλητή=μεταβλητή/παράσταση

μεταβλητή%=παράσταση	μεταβλητή%=παράσταση	μεταβλητή=μεταβλητή%παράσταση
μεταβλητή&=παράσταση	μεταβλητή&=παράσταση	μεταβλητή=μεταβλητή&παράσταση
μεταβλητή|=παράσταση	μεταβλητή|=παράσταση	μεταβλητή=μεταβλητή|παράσταση
μεταβλητή^=παράσταση	μεταβλητή^=παράσταση	μεταβλητή=μεταβλητή^παράσταση
μεταβλητή<<=x	μεταβλητή<<=x	μεταβλητή=μεταβλητή<<παράσταση
μεταβλητή>>=x	μεταβλητή>>=x	μεταβλητή=μεταβλητή>>παράσταση
μεταβλητή>>>=x	(N/A)	μεταβλητή=μεταβλητή>>>παράσταση

3.2 ΑΡΙΘΜΗΤΙΚΟΙ ΤΕΛΕΣΤΕΣ

Οι αριθμητικοί τελεστές χρησιμοποιούνται όπως στην άλγεβρα για αριθμητικές πράξεις. Προσοχή, ο τύπος Boolean απαγορεύεται να εμπλέκεται σε τέτοιους τελεστές, ενώ ο τύπος char επιτρέπεται, προφανώς λαμβανόμενος σαν υποπερίπτωση του τύπου int.

Οι βασικοί αριθμητικοί τελεστές **+**, **-**, *****, **/** χρησιμοποιούνται μεταξύ των αριθμητικών τύπων κανονικά, ενώ ο τελεστής **–** (πλην) χρησιμοποιείται και σαν τελεστής προσήμου, όπου αν x=a τότε –x=-a. Επίσης ο τελεστής **+** (συν), εκτός από την πρόσθεση αριθμητικών τιμών, χρησιμοποιείται και για την παράθεση (συνένωση) συμβολοσειρών:

string1 + string2, π.χ., "I " + "love" + " JAVA!"

Επίσης χρειάζεται προσοχή στο είδος της τιμής του αποτελέσματος των αριθμητικών τελεστών. Για παράδειγμα, αν δοθούν οι εντολές:

```
int x=2;
int y=2*x;
int z=y/3;
double w=y/3;
```

εσωτερικά θα ισχύει x = 2, y = 4, z = 1 και w = 1.333.

Ο τελεστής **%** (υπόλοιπο) ταυτίζεται με το αλγεβρικό modulo και, προσοχή, ισχύει όχι μόνο για ακεραίους αλλά και για δεκαδικούς, ενώ στη C++ ισχύει μόνο για ακεραίους (**!**):

$$\mathbf{x\ mod\ y} \overset{ορ.}{=} \mathbf{x\ \%\ y}$$

Για παράδειγμα, αν δοθούν οι εντολές:

```
int x = 20;
double y = 19.3;
Console.WriteLine(x%8 + ',' + y%8);
```

θα έχουμε στην έξοδο τις τιμές 4 και 3.3 που είναι και τα υπόλοιπα των διαιρέσεων των x και y με το 8, αντίστοιχα.

Οι τελεστές επιθεματικής αύξησης ή μείωσης **x++**, **x--** και οι τελεστές προθεματικής αύξησης ή μείωσης **++x**, **--x** αντιστοιχούν σε πράξεις αύξησης ή μείωσης κατά 1 (ένα) μεταβλητών ή εκφράσεων αριθμητικού τύπου. Αν ο τελεστής τοποθετηθεί μετά την μεταβλητή, η τιμή του x++ ή του x--, παραμένει η αρχική τιμή που είχε η μεταβλητή x. Αντίθετα, αν ο τελεστής τοποθετηθεί πριν την μεταβλητή, η τιμή του ++x ή του --x, συμπεριλαμβάνει και την αύξηση ή την μείωση της τιμής κατά 1. Εδώ, ισχύει η ακόλουθη πρόταση:

Πρόταση

Οι εκφράσεις **a = ++x**, **b = --x**, **c = x++** και **d = x--** ισοδυναμούν με τις ακόλουθες ομάδες εντολών, αντίστοιχα:

x = x +1;	**x = x - 1;**	**c = x;**	**d = x;**
a = x;	**b = x;**	**x = x + 1;**	**x = x - 1;**

Για παράδειγμα οι ακόλουθες εντολές:

```
x = 10;
y = 10;
Console.WriteLine(x++);
Console.WriteLine(++y);
```

Θα προκαλέσουν την ακόλουθη έξοδο τιμών:

```
10
11
```

3.3 ΣΧΕΣΙΑΚΟΙ ΤΕΛΕΣΤΕΣ

Οι **σχεσιακοί τελεστές (ή τελεστές σύγκρισης)** (**relational operators**) ελέγχουν αν δύο όροι ταυτίζουν τις τιμές τους ή είναι άνισοι με κάθε δυνατό τρόπο ανισότητας. Λειτουργούν από **αριστερά στα δεξιά** και έχουν συντακτικούς τύπους τους ακόλουθους:

a < b	a μικρότερο του b
a > b	a μεγαλύτερο του b
a <= b	a μικρότερο ή και ίσο του b
a >= b	a μεγαλύτερο ή και ίσο του b
a == b	a ίσο με b
a != b	a διάφορο του b

Το επιστρεφόμενο αποτέλεσμα ενός σχεσιακού τελεστή είναι true ή false, δηλαδή τύπου bool. Οι όροι a και b στην περίπτωση των τελεστών σύγκρισης για ισότητα (==, !=), μπορούν να είναι ακέραιοι, δεκαδικοί, χαρακτήρες ή και λογικοί, ενώ στην περίπτωση των τελεστών διάταξης (<, >, <=, >=), τα a, b μπορούν να είναι ακέραιοι, δεκαδικοί ή και χαρακτήρες. Για παράδειγμα, έχουμε τις ακόλουθες εντολές:

```
int x = 1;
```

```
int y = 2;
bool z = x != y;
Console.WriteLine(z);
```

θα δώσουν στην έξοδο την τιμή true. Προσοχή, οι τελεστές σύγκρισης για ισότητα, μπορούν να οδηγήσουν σε λάθη τον άπειρο Αναγνώστη. Για παράδειγμα, η έκφραση x==y!=z, προϋποθέτει πως ναι μεν τα x, y μπορούν να είναι οτιδήποτε μέσα στα πλαίσια των τελεστών αυτών, το z όμως πρέπει να είναι αναγκαστικά τύπου bool, διαφορετικά δίνεται μεταφραστικό λάθος (!), διότι η έκφραση αυτή ερμηνεύεται, από τα αριστερά στα δεξιά, με την ισοδύναμη έκφραση (x==z)!=z, οπότε το αριστερό μέρος θα δώσει true ή false, άρα και το z θα πρέπει να είναι ίδιου τύπου.

Μεγαλύτερη προσοχή, χρειάζεται για τους έμπειρους χρήστες της Java, αλλά άπειρους σε C#, οι οποίοι έχουν συνηθίσει να είναι μια υπόθεση false μόνον αν δίνει μηδενική τιμή. Για παράδειγμα, οι ακόλουθες εντολές σε C++ είναι ορθές, αλλά τελείως λανθασμένες σε C#, JAVA:

```
int x = 1;
if (x) x = 0; // Λάθος
```

Στη JAVA /C# οι εντολές αυτές θα έπρεπε να δοθούν:

```
int x = 1;
if (x != 0) x = 0;
```

Οι δύο τελεστές σύγκρισης ισότητας (==, !=) εφαρμόζονται και σε τύπους αναφοράς, όχι στις τιμές τους, αλλά αν ανήκουν στο ίδιο αντικείμενο ή όχι (instance). Η κρίσιμη αυτή παρατήρηση μπορεί να οδηγήσει σε εύλογα ερωτήματα τον Αναγνώστη, ιδιαίτερα σε περιπτώσεις σύγκρισης συμβολοσειρών (strings). Για παράδειγμα, οι ακόλουθες εντολές:

```
string s1 = "ZERO";
string s2 = "ZERO";
Console.WriteLine (s1==s2);
string s3 = new string("ZERO");
string s4 = new string("ZERO");
Console.WriteLine (s3==s4);
s3 = s4;
Console.WriteLine (s3==s4);
```

θα δώσουν σαν έξοδο τις ακόλουθες τιμές:

```
true
false
true
```

διότι τα s1, s2 είναι ψευτοαντικείμενα, ενώ για τα κανονικά αντικείμενα s3, s4, επειδή έχουν ξεχωριστά διαμερίσματα μνήμης για τις τιμές τους, ο τελεστής == θα δώσει false, διότι δε συγκρίνει τα περιεχόμενα των τιμών τους που είναι ίσα, αλλά αν είναι ταυτόσημα σαν αντικείμενα ή όχι. Όταν βέβαια στη συνέχεια

εξισωθούν, τότε το αντικείμενο s3 θα έχει το ίδιο διαμέρισμα μνήμης με το s4 και επομένως ο τελεστής == θα δώσει την τιμή true. Φιλικά συμβουλεύεται ο Αναγνώστης να χρησιμοποιεί σε συγκρίσεις τιμών strings τις ειδικές μεθόδους που παρέχει η C# (π.χ., την equals()) και όχι τους τελεστές σύγκρισης.

3.4 ΨΗΦΙΑΚΟΙ ΤΕΛΕΣΤΕΣ

Οι **ψηφιακοί (δυαδικοί) τελεστές** (**Bitwise operators**) εφαρμόζονται σε όρους τύπου **long**, **int**, **short**, **char** και **byte**, όχι άμεσα στις τιμές τους, αλλά στα bits των αντίστοιχων δυαδικών τους τιμών. Οι βασικοί ψηφιακοί τελεστές έχουν τους ακόλουθους συντακτικούς τύπους:

~a	ψηφιακό μονήρες NOT (δυαδικό συμπλήρωμα)
a & b	ψηφιακό AND (δυαδική τομή)
a | b	ψηφιακό εγκλειστικό OR (δυαδική ένωση)
a ∧ b	ψηφιακό αποκλειστικό OR (δυαδική συμμετρική διαφορά)

Ο τελεστής **~** του δυαδικού συμπληρώματος δρα στην ψηφιακή τιμή και μετατρέπει τα μηδέν σε ένα και αντίστροφα. Για παράδειγμα η ψηφιακή 8-bits τιμή του 7 είναι 00000111 και επομένως η τιμή του δυαδικού συμπληρώματος θα είναι 11111000, δηλαδή η τιμή 248. Άρα, αν δοθούν οι ακόλουθες εντολές:

```
byte a = 7;
Console.WriteLine (~a);
```

θα τυπωθεί το 248, διότι εσωτερικά η τιμή του a θα είναι 00000111. Στο σημείο αυτό υπενθυμίζεται στον Αναγνώστη πως η ψηφιακή τιμή $a_1a_2...a_m$ ισοδυναμεί με την ακόλουθη αριθμητική τιμή:

$$a_1a_2...a_m = \sum_{i=0}^{m-1} a_i 2^i$$

Για παράδειγμα η ψηφιακή τιμή 11111000 αντιστοιχεί στο άθροισμα:

$$(2^3+2^4+2^5+2^6+2^7) = 248$$

που είναι και το δυαδικό συμπλήρωμα του a του παραδείγματός μας. Επίσης χρειάζεται προσοχή στις αρνητικές τιμές όπου η C# χρησιμοποιεί για τις δυαδικές τους τιμές τον κώδικα του συμπληρώματος ως προς 2 (two's complement). Αυτό σημαίνει πως παίρνεται η αντίστοιχη ψηφιακή τιμή, εφαρμόζεται εσωτερικά ο τελεστής του δυαδικού συμπληρώματος και στη συνέχεια προστίθεται το 1. Για παράδειγμα, το -7 έχει σαν δυαδική τιμή το συμπλήρωμα της 00000111, δηλαδή την 11111000 και προστίθεται το 1, οπότε προκύπτει η τιμή 11111010. Για την αποκωδικοποίηση του -7, πρώτα παίρνεται το δυαδικό συμπλήρωμα της τιμής 11111010, οπότε προκύπτει η τιμή 00000101 και στη συνέχεια προστίθεται το 1, οπότε προκύπτει τελικά 00000111 που αντιστοιχεί στην τιμή 7.

ΠΙΝΑΚΑΣ ΟΡΙΣΜΟΥ ΤΙΜΩΝ ΨΗΦΙΑΚΩΝ ΤΕΛΕΣΤΩΝ

A	**B**	**a|b**	**A&b**	**a∧b**	**~a**
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

Το **ψηφιακό AND** εκφράζεται με τον τελεστή **&** και δίνει τιμή που αντιστοιχεί στον δυαδικό αριθμό που προέρχεται από την πράξη με τον ακόλουθο προτασιακό τύπο:

«Αν το bit i της δυαδικής τιμής του a είναι 1 και το bit i της δυαδικής τιμής του b είναι 1, τότε το bit i του αποτελέσματος της πράξης a&b είναι 1, διαφορετικά, σε κάθε άλλη περίπτωση είναι 0. Η εφαρμογή της πρότασης για κάθε i δίνει τον τελικό επιστρεφόμενο δυαδικό αριθμό».

Για παράδειγμα αν ισχύει:

A = 53 δυαδική τιμή = 00110101

B = 102 δυαδική τιμή = 01100111

τότε προκύπτει το ακόλουθο αποτέλεσμα με τον τελεστή &:

a&b = 00100101

που αντιστοιχεί στην αριθμητική τιμή 37.

Το **ψηφιακό εγκλειστικό OR**, εκφράζεται με τον τελεστή **|** και δίνει τιμή που αντιστοιχεί στον δυαδικό αριθμό που προέρχεται από την πράξη με τον ακόλουθο προτασιακό τύπο:

«Αν το bit i της δυαδικής τιμής του a είναι 0 και το bit i της δυαδικής τιμής του b είναι 0, τότε το bit i του αποτελέσματος της πράξης a|b είναι 0, διαφορετικά, σε κάθε άλλη περίπτωση είναι 1. Η εφαρμογή της πρότασης για κάθε i δίνει τον τελικό επιστρεφόμενο δυαδικό αριθμό».

Για παράδειγμα, αν a = 53 και b = 102, τότε ισχύει:

a|b = 01110111, που αντιστοιχεί στον αριθμό 119.

Το **ψηφιακό αποκλειστικό OR**, εκφράζεται με τον τελεστή ∧ και δίνει τιμή που αντιστοιχεί στον δυαδικό αριθμό που προέρχεται από την πράξη με τον ακόλουθο προτασιακό τύπο:

«Αν το bit της δυαδικής τιμής του a είναι ταυτόσημο με το bit i της δυαδικής τιμής του b, τότε το bit i του αποτελέσματος της πράξης a∧b είναι 0, διαφορετικά, σε κάθε άλλη περίπτωση είναι 1. Η εφαρμογή της πρότασης για κάθε i δίνει τον τελικό επιστρεφόμενο δυαδικό αριθμό».

Για παράδειγμα αν a = 53 και b = 102, τότε ισχύει:

a∧n = 01010010, που αντιστοιχεί στον αριθμό 92

Αν το bit i υπάρχει (1) ή όχι (0), στο στοιχείο i κάποιου συνόλου αναφοράς (π.χ. του συνόλου ~0), τότε υπάρχει η ακόλουθη αντιστοιχία των βασικών ψηφιακών τελεστών με τις πράξεις των συνόλων:

~	Άρνηση
&	Τομή
|	Ένωση
∧	συμμετρική διαφορά (ένωση πλην την τομή)

Στη συνέχεια δίνεται ένα απλό παράδειγμα εκπαιδευτικού επιπέδου κατανόησης πράξεων με bits:

```
short i = 2, j = 5;
short z = (i&j)|(i∧j);
Console.WriteLine ("z =" + z);
```

Στο παράδειγμά μας, δηλώνονται οι short ακέραιοι i και j με τιμές 2 και 5 αντίστοιχα, που αντιστοιχούν στις δυαδικές τιμές;

0000000000000010 0000000000000101

Η δεύτερη εντολή, θα επιστρέψει μία short ακέραια τιμή στη μεταβλητή z. το δεξιό μέλος της εντολής θα εκτελεστεί με βάση τον τελεστή |, από αριστερά στα δεξιά. Άρα θα εκτελεστεί πρώτα η πράξη (i & j), η οποία δίνει:

0000000000000000

Στη συνέχεια, θα εκτελεσθεί η πράξη (i ∧ j), η οποία δίνει:

0000000000000111

Τελικά, η πράξη των δύο αυτών τιμών με τον τελεστή |, θα επιστρέψει σαν τιμή:

0000000000000111

η οποία αντιστοιχεί στην αριθμητική τιμή 7. Επομένως η έξοδος των εντολών αυτών θα έχει τη μορφή:

```
z = 7
```

3.5 ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ

Οι **λογικοί τελεστές** (**boolean logical operators**) λειτουργούν αποκλειστικά μεταξύ όρων τύπου **bool**. Το αποτέλεσμα είναι επίσης bool και η δράση των τελεστών είναι από τα αριστερά στα δεξιά. Οι βασικοί λογικοί τελεστές έχουν τους ακόλουθους συντακτικούς τύπους:

a&b	λογικό AND
a|b	λογικό εγκλειστικό OR
a∧b	λογικό αποκλειστικό OR (XOR)
!a	λογική άρνηση (πράξη NOT)
a&&b	λογικό AND σύντομης εκτίμησης
a||b	λογικό OR σύντομης εκτίμησης

Οι τελεστές &, | και ∧ λειτουργούν ακριβώς όπως οι αντίστοιχοι ψηφιακοί τελεστές όπου οι τιμές των bits 0 και 1, αντικαθίστανται με τις λογικές τιμές false και true, αντίστοιχα. Ο Αναγνώστης παραπέμπεται στην προηγούμενη παράγραφο για μια πλήρη ανάλυση αυτών.

Ο τελεστής **!** αναφέρεται στη **λογική άρνηση** (**NOT**), με αποτέλεσμα λογικά αντίθετο από εκείνο της έκφρασης που εφαρμόζεται. Για παράδειγμα, η εντολή:

```
if(!e) e=true;
```

ισοδυναμεί με τον προτασιακό αλγεβρικό τύπο «αν το e είναι false, να τεθεί το e true».

Ο τελεστής **&&** αναφέρεται στο λογικό AND (**logical AND**), αλλά με σύντομη εκτίμηση (**short-circuit**) σε αντίθεση με τον τελεστή **&** που πραγματοποιεί πλήρη εκτίμηση. Συγκεκριμένα ισχύει ο αναλυτικός αλγόριθμος που παρατίθεται, όπου αν η τιμή του a είναι false, η όλη έρευνα σταματά με επιστρεφόμενη τιμή το false. Αυτό είναι και το κύριο χαρακτηριστικό της τακτικής της σύντομης εκτίμησης.

Αναλυτικός Αλγόριθμος Υπολογισμού

y = a&&b

```
Αν a = false, θέσε
    y = false
διαφορετικά
    Αν b = false, θέσε
        y = false
    διαφορετικά
        y = true
```

Ο τελεστής **||** αναφέρεται στο λογικό OR (**logical OR**), το οποίο είναι εγκλειστικό και σύντομης εκτίμησης (**short-circuit**), σε αντίθεση με τον τελεστή **|** που πραγματοποιεί πλήρη εκτίμηση. Συγκεκριμένα, ισχύει ο αναλυτικός αλγόριθμος που παρατίθεται, όπου αν η τιμή του a είναι true, η όλη έρευνα σταματά με επιστρεφόμενη τιμή το true χωρίς καμία άλλη εξέταση (σύντομη εκτίμηση).

Αναλυτικός Αλγόριθμος Υπολογισμού

y = a||b

Αν a = true, θέσε

y = true

διαφορετικά

Αν b = true, θέσε

y = true

διαφορετικά

y = false

Παραδείγματα εκφράσεων που εμπλέκουν λογικούς τελεστές είναι τα ακόλουθα:

```
if((x==y)&&(z>=3.33)) w=1; //1
if((p==0)||(b[i]>b[j])) w=1; //1
if((e&&f)||(r!=g(k,z)) w=1; //1
```

Στο παράδειγμα 1, αν η τιμή του x ταυτίζεται με την τιμή του y, τότε μόνον θα εξετασθεί αν η τιμή του z είναι μεγαλύτερη ή και ίση με την τιμή 3.33, οπότε και θα τεθεί στο w η τιμή 1. Αντίθετα, στο παράδειγμα 4 θα γίνει πλήρης εκτίμηση. Θα εξετασθεί και αν οι τιμές των x, y ταυτίζονται και αν η τιμή του z είναι μεγαλύτερη ή ίση από το 3.33, σε κάθε περίπτωση:

```
if((x==y)&(z>=3.33)) w=1; //4
```

Στο παράδειγμα 2, αν το p είναι μηδέν, θα τεθεί w=1 χωρίς καμία άλλη έρευνα. Διαφορετικά, θα εξετασθεί αν b[i]>b[j], οπότε και πάλι θα τεθεί w=1. Διαφορετική συμπεριφορά θα έχουν τα παραδείγματα 5 και 6:

```
if((p==0)|(b[i]>b[j])) w=1; //5
if((p==0)^(b[i]>b[j])) w=1; //6
```

όπου στο παράδειγμα 5 θα γίνει πλήρης έρευνα και αν ισχύει p==0 και αν ισχύει b[i]>b[j]. Αν τύχει να ισχύει ένα από τα δύο τουλάχιστον, τότε θα τεθεί w=1. Στο παράδειγμα 6, θα γίνει και πάλι η πλήρης αντίστοιχη έρευνα και αν ισχύει μία ακριβώς από τις δύο υποθέσεις, τότε μόνον θα τεθεί w=1.

Στο παράδειγμα 3, αν e και f είναι true, θα τεθεί w=1. Διαφορετικά, θα εξετασθεί αν η τιμή του r είναι διάφορη της τιμής που επιστρέφει η μέθοδος g. Αν ναι, θα τεθεί w=1.

Συμπερασματικά, οι λογικοί τελεστές **&**, **&&** διαφέρουν στο ότι ο πρώτος εκτός του ότι εφαρμόζεται και σε πράξεις με bits, όταν εφαρμοσθεί σε δύο όρους με τιμές bool, εκτελεί **πλήρη εκτίμηση** και μετά συμπερασματολογεί. Αντίθετα, ο τελεστής **&&** προβαίνει στις **ελάχιστες δυνατές εκτιμήσεις** για να έχει συμπέρασμα. Ακριβώς, αντίστοιχα ισχύουν και για τους τελεστές **|**, **||**. Ισχύει η ακόλουθη πρόταση:

Πρόταση

Ο αναμενόμενος υπολογιστικός χρόνος των τελεστών &&, || είναι μικρότερος ή ίσος από τους τελεστές &, | σε ίδιους bool όρους.

Η απόδειξη της πρότασης είναι προφανής διότι οι τελεστές &&, || λειτουργούν με την μέθοδο της σύντομης εκτίμησης, σε αντίθεση με τους &, | που ακολουθούν την τακτική της πλήρους εκτίμησης. Ωστόσο, υπάρχουν περιπτώσεις που οι αναμενόμενοι υπολογιστικοί χρόνοι ταυτίζονται σε κάθε περίπτωση. Για παράδειγμα:

```
bool x = true;
bool y = false;
bool z = (x&y)|(x&!y);
bool w = (x&&y)||(x&&!y);
```

οι τιμές των z, w ταυτίζονται (true) καθώς επίσης και οι αντίστοιχοι υπολογιστικοί χρόνοι εκτέλεσης υπολογισμού των τιμών τους... Αντίθετα, αν δοθούν επιπρόσθετα και οι εντολές:

```
bool t = (!x&y)|(x&!y);
bool f = (!x&&y)||(x&&!y);
```

τότε ο υπολογιστικός χρόνος υπολογισμού του f θα είναι κατά πολύ μικρότερος εκείνου του υπολογισμού του t, ενώ οι τιμές των t, f θα είναι επίσης true.

3.6 ΤΕΛΕΣΤΕΣ ΕΚΧΩΡΗΣΗΣ

Ο όρος **εκχώρηση** εννοεί την ανάθεση τιμής σε μία μεταβλητή. Όταν η τιμή μιας έκφρασης ή μεταβλητής ή και σταθεράς, σημειώνεται σαν τιμή σε μια μεταβλητή, τότε γίνεται συνήθως, λόγος για πράξη αντικατάστασης (**Assignment**). Προσοχή, στις γλώσσες προγραμματισμού η έννοια της αντικατάστασης εκφράζεται κατά κανόνα με εντολή (statement) και όχι με τέλεση. Η C++ χρησιμοποίησε πρώτη τελεστές εκχώρησης, τους οποίους κληρονόμησε και στην JAVA, C#. Οι τελεστές αυτοί ανήκουν στην κατηγορία των **Ειδικών Τελεστών** που περιέχουν τον τελεστή της απλής (όχι λογικής) ισότητας (=). Ο συντακτικός τύπος μιας τέτοιας πράξης, έχει τη μορφή:

μεταβλητή τελεστής = παράσταση (1)

όπου **υπολογίζεται η τιμή της παράστασης** με βάση τον **τελεστή** που σημειώνεται και στη συνέχεια τίθεται το αποτέλεσμα σαν τιμή στη **μεταβλητή** του αριστερού μέλους που δίνεται. Είναι φανερό, πως η όλη πράξη της αντικατάστασης, εκτελείται **από τα δεξιά προς τα αριστερά**, ακόμη και σε πολύπλοκες καταστάσεις. Η δε τιμή του δεξιού μέλους, μετατρέπεται σε είδος τιμής του αριστερού μέλους, προφανώς όχι πάντα με επιτυχία. Όταν το διαμέρισμα μνήμης για την μεταβλητή της όλης πράξης, έχει λιγότερα bits από την τιμή του δευτέρου μέλους, τότε θα πρέπει να εφαρμοσθεί ειδικός τελεστής μετατροπής, ο οποίος θα αναλυθεί σε μια επόμενη παράγραφο.

Οι βασικοί τελεστές εκχώρησης, πλην εκείνων που περιλαμβάνουν περιστροφή και κύλιση (με bits), είναι οι ακόλουθοι:

a=b	Τελεστής αντικατάστασης
a+=b	A=a+b
a-=b	A=a–b
a*=b	A=a*b
a/=b	A=a/b
a%=b	A=a%b
a&=b	A=a&b
a|=b	A=a|b
a∧=b	A=a∧b

όπου γενικά, ισοδυναμούν με τον ακόλουθο συντακτικό τύπο:

μεταβλητή = μεταβλητή τελεστής παράσταση (2)

Πρόταση

Οι συντακτικοί τύποι (1) και (2) από πλευράς αποτελέσματος είναι ισοδύναμοι, αλλά η αποτίμηση της μεταβλητής στην (1) γίνεται μόνον μία φορά.

Για παράδειγμα αν δοθεί η εντολή:

```
x + = y;
```

αυτή ισοδυναμεί με την ακόλουθη εντολή:

```
x = x + y;
```

Ωστόσο, οι δύο αυτές εντολές δεν είναι απόλυτα ισοδύναμες, διότι η δεύτερη έχει πολύ μεγαλύτερο υπολογιστικό χρόνο από την πρώτη.

Στη C# και τη JAVA, ο απλός τελεστής εκχώρησης μπορεί να περιλαμβάνει περισσότερα από ένα αριστερά μέλη. Σε μια τέτοια περίπτωση η ισότητα αντικατάστασης υπακούει στον ακόλουθο συντακτικό τύπο:

μεταβλητή1 = μεταβλητή2 = ... = μεταβλητήk = έκφραση;

όπου αρχικά θα υπολογισθεί η τιμή της **έκφρασης**, στη συνέχεια θα δοθεί η τιμή αυτή (ίσως μετά από μετατροπή) στην **μεταβλητήk**, κλπ.. μέχρις ότου δοθεί και στην **μεταβλητή1**. Για παράδειγμα, αν δοθούν οι εντολές:

```
int a, b, c, d;
a = b = c = d = 10;
```

θα δοθεί σταδιακά η τιμή 10, στις μεταβλητές d, c, b και a.

3.7 ΤΕΛΕΣΤΕΣ ΚΥΛΙΣΗΣ

Οι τελεστές κίνησης (κύλισης ή ολίσθησης) εφαρμόζονται σε μια τιμή a αφού γίνει μετατόπιση στη δυαδική του τιμή, τόσα bits, όσα η τιμή του b. Οι τελεστές κύλισης είναι οι ακόλουθοι:

a<<b	αριστερή κύλιση των bits του a κατά b θέσεις
a>>b	δεξιά κύλιση των bits του a κατά b θέσεις
a<<=b	αριστερή κύλιση με εκχώρηση τιμής a=a<<b
a>>=b	δεξιά κύλιση με εκχώρηση τιμής a=a>>b

Μια πρώτη βασική παρατήρηση, είναι ότι οι τελεστές ολίσθησης, έχουν νόημα σε περιπτώσεις που ο όρος a είναι τύπου **int** ή **long**, οπότε η απόσταση κύλισης μπορεί να είναι **από μηδέν μέχρι τριάντα ένα** (0-31) ή **από μηδέν μέχρι εξήντα τρία** (0-63), αντίστοιχα. Βέβαια, η τιμή της απόστασης, ταυτίζεται με την τιμή της παραμέτρου b.

Ο τελεστής της αριστερής κύλισης **a<<b**, προκαλεί ολίσθηση των bits της δυαδικής τιμής του a κατά b θέσεις προς τα αριστερά. Τα bits της τιμής που παίρνουν την θέση 31 (τύπου int) ή τη θέση 63 (τύπου long) στην ουσία χάνονται, ενώ τα bits που έρχονται από το δεξιό τέλος της τιμής του a είναι όλα μηδέν. Για παράδειγμα, αν ισχύει:

```
int a=7;
int c=a<<3;
```

τότε, η τιμή του c, θα είναι 56, διότι η πλήρης δυαδική τιμή του a είναι:

00000000000000000000000000000111

η οποία μετά την ολίσθηση στα αριστερά κατά τρεις θέσεις, θα γίνει:

00000000000000000000000000111000

η οποία ισοδυναμεί με το 56 που θα δοθεί στην μεταβλητή c.

Ο τελεστής της δεξιάς κύλισης **a>>b** προκαλεί ολίσθηση των bits της δυαδικής τιμής του a κατά b θέσεις προς τα δεξιά. Τα bits που περνούν το τέλος της τιμής χάνονται, ενώ τα bits που προέρχονται σαν νέα από τα αριστερά, διατηρούν την τιμή του top bit, προφανώς για την **διατήρηση προσήμου** της τιμής του a. Για παράδειγμα, αν ισχύει:

```
int a=35;
int b=a>>3;
```

τότε η δυαδική τιμή του a θα είναι:

00000000000000000000000000100011

ενώ μετά τη δεξιά ολίσθηση, η δυαδική τιμή του b θα είναι:

00000000000000000000000000000100.

Τέλος, για τους τελεστές ολίσθησης με ανάθεση τιμής, απλά το αποτέλεσμα που προκύπτει μετά την ολίσθηση δίνεται στην μεταβλητή της ολίσθησης ξανά. Για παράδειγμα οι ακόλουθες εντολές:

```
int x=2;
int y=3;
x>>=1;
y<<=1;
```

θα δώσουν τελικά στα x,y τις τιμές 1 και 6 αντίστοιχα.

3.8 Ο ΥΠΟ ΣΥΝΘΗΚΗ ΤΕΛΕΣΤΗΣ

Η Java κληροδότησε στη C# έναν **τριαδικό (ternary)** τελεστή, τον, **υπό συνθήκη τελεστή (conditional operator)**, ο οποίος έχει τον ακόλουθο συντακτικό τύπο:

Υπόθεση Α? Εκφραση Β: Εκφραση Γ

Όπου, αν η τιμή της υπόθεσης Α είναι true, εκτελείται μόνο η έκφρασηΒ, διαφορετικά εκτελείται η έκφραση Γ. Μια ακριβώς έκφραση θα εκτελεστεί, της οποίας η τιμή επιστρέφεται σαν τιμή της όλης πράξης. Για παράδειγμα, η έκφραση:

```
(2+1!=4)?117:x++
```

έχει την υπόθεση του πρώτου ορίσματος true, πράγμα που οδηγεί στη τιμή 117, χωρίς να αυξάνεται η τιμή του x κατά ένα. Ο υπό συνθήκη τελεστής, είναι ιδανικός σε περιπτώσεις δίτιμης λογικής. Για παράδειγμα η αλγεβρική έκφραση:

minab =min(a,b), ισοδυναμεί με την εντολή:
minab =a<b? a:b;

όπου τελικά ο μικρότερος μεταξύ των μεταβλητών a,b θα δοθεί σαν τιμή στην minab

Στον υπό συνθήκη τελεστή, οι εκφράσεις Β και Γ επιστρέφουν τον ίδιο τύπο τιμής και απαγορεύεται να είναι τύπου void. Τέλος, υπενθυμίζεται η σχέση ισοδυναμίας του υπό συνθήκη τελεστή και της απλής εντολής if. Είναι πραγματικά κρίμα, που οι εκφράσεις Β και Γ του υπό συνθήκη τελεστή, πρέπει να είναι απλές εκφράσεις (expressions) που να δίνουν κάποια τιμή. Διαφορετικά θα μπορούσε ο τελεστής αυτός να αντικαταστήσει ένα μεγάλο σύνολο εντολών if

3.9 ΕΙΔΙΚΟΙ ΤΕΛΕΣΤΕΣ

Τόσο η Java όσο και η C# περιέχουν και τρεις τελεστές που ανήκουν σε **ειδική κατηγορία τελεστών (specific operators)**. Μάλιστα, οι πρώτοι δύο έχουν μεγάλη προτεραιότητα εκτέλεσης:

1. **Δέσμευση μνήμης για αντικείμενο**
2. **Ρητή μετατροπή τύπου**
3. **Απόφαση για αν ένα αντικείμενο ανήκει σε μια κλάση**

Η Java αλλά και η C# προβαίνουν σε **αυτόματη μετατροπή τιμών (automatic conversion)** μόνο σε περιπτώσεις που οι διαφορετικές σε είδος τιμές είναι συμβατές. Δύο τύποι δεδομένων είναι συμβατοί αν το διαμέρισμα τιμών της τιμής του τελικού τύπου είναι μεγαλύτερο ή και ίσο από το αντίστοιχο του αρχικού τύπου. Για παράδειγμα:

```
int x= 123456789;
```

```
long y=x;
```

η μεταβλητή y θα έχει τελικά την ίδια τιμή με τη x, αλλά μετασχηματισμένη από 32bits σε 64 bits, δηλαδή την τιμή 123456789L. Όταν όμως, δύο τύποι δεν είναι συμβατοί τοτε πρέπει να εφαρμοστεί ο **τελεστής ρητής μετατροπής** για μια απόπειρα επιτυχούς μετατροπής, με βάση τον συντακτικό τύπο:

τελικος_τύπος τελική_τιμή=(τελικός_τύπος) παλαιου_τύπου_τιμή

για παράδειγμα:

int x=117;
byte y=(byte)x;

Η τιμή 117 θα είναι τελικά η τιμή του y, σε 8bits πεδίου αντί σε 32bits που ήταν η ίδια τιμή στο διαμέρισμα της μεταβλητής x.

Στη Java και στη C# η δέσμευση μνήμης για ένα αντικείμενο γίνεται στη διάρκεια της εκτέλεσης με τη βοήθεια του τελεστή **new.** Ο γενικός συντακτικός τύπος της εντολής της δέσμευσης μνήμης είναι:

Όνομα_κλάσης Αντικείμενο_κλάσης =new Όνομα_κλάσης (λίστα_αρχικών_τιμών)

Όπου η λίστα αρχικών τιμών αναφέρεται στις αρχικές τιμές που θα έχουν τα μέλη του αντικειμένου που θα περάσουν μέσα από την κλήση αντίστοιχης κατασκευαστικής μεθόδου της κλάσης. Όταν ένα αντικείμενο κλάσης οριστεί, δεν έχει ακόμη αποκτήσει διαμέρισμα μνήμης για τις τιμές των μελών του, μέχρις ότου γίνει χρήση του τελεστή new. Για παράδειγμα αν δοθούν οι εντολές:

```
class zero{
int x;
double y;
byte z;}

class demo{
public static void main (String [] args){
zero object=new zero();
object.x=114;
object.x=117.114;
object.z=(byte)object.x;}}
```

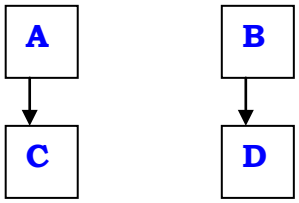
θα γίνει δέσμευση μνήμης για το αντικείμενο object, όπου θα αποθηκευτούν οι τιμές των μελών του x, y, z σαν 114, 117.114, 114 αντίστοιχα σε έκταση (32 +64+8) bits.

Υπάρχουν περιπτώσεις που θέλουν μια επιβεβαίωση για το αν ένα αντικείμενο ανήκει σε μια κλάση ή όχι. Για παράδειγμα, αν έχουμε την κλάση A και τις δύο υποκλάσεις X και Y, τότε ένα αντικείμενο του X μπορεί να είναι και του A ή ένα αντικείμενο του Y θα είναι και του A, αλλά είναι λάθος ένα αντικείμενο του X να θεωρείται αντικείμενο και του Y και αντίθετα. Η Java

προσφέρει τον ειδικό τελεστή **instanceof** για μια απάντηση στο ανωτέρω ερώτημα στη διάρκεια της εκτέλεσης. Η C# χρησιμοποιεί τον ειδικό τελεστή **is** με γενικό συντακτικό τύπο:

αντικείμενο is κλάση

Όπου αν το αντικείμενο ανήκε στην κλάση που δίδεται, η όλη έκφραση επιστρέφει την τιμή true, διαφορετικά επιστρέφει την τιμή false. Για παράδειγμα αν έχουμε την ακόλουθη κληρονομικότητα κλάσεων και υποκλάσεων:




```
A a=new A();
B b=new B();
C c=new C();
D d=new D();
```

Και δοθούν επιπρόσθετα οι εντολές:

```
bool x1, x2,x3,x4,x5,x6,x7;
x1=a is A;
x2=b is B;
x3=c is C;
x4=d is D;
x5=c is A;
x6=d is B;
x7=c is B;
```

μόνο η μεταβλητή x7 θα πάρει την τιμή false,διότι είναι βέβαια αδύνατον το αντικείμενο c της κλάσης C να είναι σε θέση να εκπροσωπήσει **στιγμιαίες εικόνες τιμών** των μελών της κλάσης B.

3.10 ΑΣΚΗΣΕΙΣ

1. Να γραφεί πρόγραμμα σε C#, το οποίο να υπολογίζει και να εμφανίζει το άθροισμα των στοιχείων μιας μήτρας ακεραίων 50x40, να υπολογίζει και να εμφανίζει το μέγιστο των άρτιων γραμμών και το ελάχιστο των περιττών γραμμών.
2. Να εκτελεσθεί ο παρακάτω κώδικας και να ερμηνευτούν οι διαφορές στις δυο δομές επανάληψης.

```
using System;

namespace Csoper
{
    class Class1
```

```
  {    [STAThread]
       static void Main(string[] args)
        {  int i;
           int j;
           for(i=1;i<10;i++)
           {  System.Console.WriteLine("I="+i);
              i=i++;
           }
           for(j=1;j<10;j++)
           {
             System.Console.WriteLine("J="+j);
             j=++j;
           }
           System.Console.ReadLine();
        }
  }
}
```

ΚΕΦΑΛΑΙΟ 4ο

4. ΕΝΤΟΛΕΣ

4.1 ΕΙΣΑΓΩΓΗ

Οι βασικές διαφορές JAVA και C# στο θέμα των εντολών είναι ότι αφενός η μεν C# έχει δυο επιπρόσθετες εντολές, την εντολή **foreach** και την εντολή **goto,** αφετέρου η C# έχει τις απλές εντολές **Continue και Break χωρίς label** όπως αντίθετα έχει η JAVA. Προφανώς αντικαθιστά τα σύνθετα break και continue της JAVA με το goto ενώ με το foreach έχει την δυνατότητα επαναληπτικής διαδικασίας με βάση τα στοιχεία ενός πεπερασμένου συνόλου (συλλογές). Ειδικότερα τόσο η JAVA όσο και η C# έχουν όλες εκείνες τις **εντολές (statements)** για τον ορθό και ταχύ **έλεγχο ροής εργασιών (control flow),** όπως **επαναληπτικές διαδικασίες (loops), εντολές επιλογής και απόφασης**, καθώς και **εντολές μεταφοράς ελέγχου**.
Οι βασικές εντολές και οι διαφορές τους επισημαίνονται στον παρακάτω πίνακα:

Java	C#	Σχόλια
If…else	If…else	Εντολή συνθήκης.
Switch…case	switch…case	Εντολή συνθήκης.
For	For	Επαναλαμβανόμενο loop
Do	Do	Loop μετά τη συνθήκη.
While	While	Loop πριν τη συνθήκη.
-	Foreach	Συλλογή συγκεκριμένων απαριθμήσεων.
-	Goto	Στην Java παρέχεται η εντολή goto αλλά δε χρησιμοποιείται. Κάνει άλμα στην ετικέτα (label) χωρίς συνθήκη.
continue(label)	continue	Ξεκινάει μια νέα επανάληψη του loop.
Break(label)	Break	Τερματισμός του loop.
Return	Return	Επιστροφή από ένα λειτουργικό μέλος

Ακόμη εντολές θεωρούνται το **compound statement (λογισμική παράγραφος)**, η **κενή εντολή** και η **εντολή ισότητα αντικατάστασης (έμμεση ή άμεση) (expression statement)**.

Τέλος υπάρχουν και οι **ειδικές εντολές χειρισμού των εξαιρέσεων (exception handling statements)** με κυριότερες το **try**, **catch** και άλλες που θα αναλυθούν εκτενέστερα σε επόμενο κεφάλαιο.

Οι εντολές των αντικειμενοστρεφών γλωσσών προγραμματισμού συγκροτούνται από λέξεις κλειδιά (keywords), τελεστές, ειδικά σύμβολα, σταθερές και μεταβλητές. Μερικές από αυτές δέχονται επικράτεια εντολών (body), όπου μπορούν να υπάρχουν άλλες εντολές με επικράτεια ή όχι (φαινόμενο nesting). Εντολές επίσης θεωρούνται οι εντολές δηλώσεων αντικειμένων, μεταβλητών, και αρχικών τιμών. Ιδιαίτερη προσοχή στις σημαντικές διαφορές ανάμεσα σε JAVA και C# στην χρήση των εντολών break, continue, η απουσία του goto από την πρώτη, αλλά και η σημαντική στέρηση της JAVA σε εντολές (I/O), όπου τα πάντα γίνονται μέσα από παρασχόμενες

μεθόδους συνοδευτικών κλάσεων. Αυτά τα σημαντικά κενά καλύπτει πολύ πετυχημένα η C#.

4.2 ΕΝΤΟΛΕΣ ΚΑΙ ΕΚΦΡΑΣΕΙΣ

Κάθε έκφραση είναι μια απλή εντολή, που όταν εκτελείται παράγει κάποιο αποτέλεσμα. Με την προσθήκη όμως του ειδικού τελεστή **;** στο τέλος μιας έκφρασης, αυτή μετατρέπεται σε εντολή. Συχνά, σε μια τέτοια εντολή, υπάρχει και αριστερό μέλος, μια μεταβλητή στην οποία καταχωρείται η προκύπτουσα τιμή της δεξιάς έκφρασης, αφού βέβαια γίνει κάποια μετατροπή αυτής, αν αυτό απαιτείται. Η εντολή αυτή καλείται **εντολή αντικατάστασης (assignment statement)** και έχει τον γενικό συντακτικό τύπο:

Όνομα_μεταβλητής = έκφραση;

όπου η τιμή της έκφρασης θα καταχωρηθεί στο πεδίο τιμών στην κύρια μνήμη της μεταβλητής του αριστερού μέρους. Τα ακόλουθα παραδείγματα είναι απλές εντολές αντικατάστασης.

```
x++;
y=(y+x-2);
a.f(y);
```

4.3 Η ΕΝΤΟΛΗ IF…ELSE

Η εντολή if…else χρησιμοποιείται για λογικές αποφάσεις μέσα στην ροή του προγράμματος. Ο γενικός συντακτικός τύπος είναι ο ακόλουθος :

```
If (condition)
{
Επικρατεια..1;
}
Else
{
Επικράτεια…2;}
```

όπου αν ισχύει η Υπόθεση του if εκτελείται μόνο η Επικράτεια 1, διαφορετικά μόνο η Επικράτεια 2.Το else δεν είναι υποχρεωτικό και συχνά στην πράξη απουσιάζει. Οι εντολές επικράτειας όπως και στην JAVA βρίσκονται μέσα σε άγκιστρα {…}.Αν η επικράτεια αποτελείται από μια μόνο εντολή τότε τα άγκιστρα είναι προαιρετικά. Ιδιαίτερη προσοχή χρειάζεται στην περίπτωση των πολλαπλών if (nested ifs) καθώς και στην ανυπαρξία που παρουσιάζουν οι γλώσσες αντικειμενοστρεφούς προγραμματισμού στην εντολή elseif. Ένα κενό που καλύπτεται κατασκευαστικά με την ακόλουθη δομή:

```
If (condition)
{
Επικρατεια..1;
}
Else if
{
Επικράτεια…2;}
```

Ένα απλό παράδειγμα εκτέλεσης της εντολής if είναι το ακόλουθο:

```
using System;
  class iftest
   {
    static void Main(string[] args)
    { string MyName = "Periklis";
      string herName = "Aspasia";
      Bool married = true;
      Console.WriteLine("My name is " + MyName);
       if(married==true)
        Console.WriteLine("and my wife is" + herName + ".");
     }
    }
```

4.4 ΕΠΑΝΑΛΗΠΤΙΚΕΣ ΔΙΑΔΙΚΑΣΙΕΣ

Μια **επαναληπτική διαδικασία (iteration statement)** συγκροτεί μια σύνθετη εντολή, η οποία επαναλαμβάνει εκτελεστικά το κύριο σώμα των εντολών που περιέχει, μέχρις ότου μια **συνθήκη τερματισμού** ικανοποιηθεί ή κάποια **ειδική εντολή απόδρασης (escape statement)** εκτελεσθεί όποτε και περατούται η όλη επαναληπτική διαδικασία. Οι επαναληπτικές διαδικασίες ονομάζονται απλά επαναλήψεις (loops) και είναι στην ουσία ο πρωταγωνιστής της πρώτης νόρμας των προγραμμάτων, δηλαδή του αναμενόμενου υπολογιστικού χρόνου.

Στις αντικειμενοστραφής γλώσσες προγραμματισμού οι επαναλήψεις περιέχουν τρεις βασικές ιδιότητες.

- **Αρχική τιμή διαδικασίας (αρχικοποίηση)**, όπου δίνονται αρχικές τιμές σε ορισμένες μεταβλητές για να αρχίσουν οι επαναλήψεις.
- **Έλεγχος επανάληψης (διακόπτης)** όπου κάποια συνθήκη ή και περισσότερες ορίζουν αν θα σταματήσουν οι επαναλήψεις.
- **Μεταβολή για την επόμενη επανάληψη**, όπου προσδιορίζονται οι μεταβολές που θα ισχύουν για την επόμενη επανάληψη.
- **Αναγκαστική πρώτη εκτέλεση**, αν θα εκτελεστεί αναγκαστικά την πρώτη φορά το σώμα εκτέλεσης των εντολών της σύνθετης επαναληπτικής διαδικασίας.

ΕΝΤΟΛΗ	Αρχικοποίηση	Διακόπτης	Μεταβολή	Πρώτη εκτέλεση
While	Όχι	Αρχή	Όχι	Όχι
Do-while	Όχι	Τέλος	Όχι	Ναι
For	Ναι	Αρχή	Ναι	Όχι

ΘΕΩΡΗΜΑ

Οι εντολές while, do-while, for είναι κατασκευαστικά ισοδύναμες μεταξύ τους, όπου κάθε μια από τις εντολές μπορεί να εκφραστεί ισοδύναμα με οποιαδήποτε από τις υπόλοιπες.

ΠΡΟΤΑΣΗ

Αν ο διακόπτης μιας επαναληπτικής διαδικασίας είναι προγραμματιστικά ανεξάρτητος από τις υπόλοιπες εντολές αυτής, ενώ στο σώμα της διαδικασίας δεν υπάρχει εντολή απόδρασης, τότε υπάρχει κεντρικό σημείο αυτοκατάρρευσης του προγράμματος, συστολικής μορφής.

Η απόδειξη της πρότασης είναι εμφανής. Εφόσον ο έλεγχος επανάληψης είναι προγραμματιστικά ανεξάρτητος, έπεται ότι από επανάληψη σε επανάληψη δεν επηρεάζεται από κάποια εντολή, με άμεση συνέπεια τον εγκλωβισμό της όλης διαδικασίας, χωρίς καμία ελπίδα εξόδου και με παράλληλη αδυναμία του κώδικα για κάποιο σχετικό τερματισμό ή μήνυμα. Τελικά ο επεξεργαστής συστολικά φυλακίζεται μέσα σε ατελείωτες επαναλήψεις στο σώμα εντολών της επαναληπτικής διαδικασίας. Βέβαια, η πρόταση ισχύει αν δεν υπάρχει εντολή απόδρασης μέσα στο σώμα της επαναληπτικής διαδικασίας, η οποία κάτω από ορισμένες συνθήκες έχει την δυνατότητα απεγκλωβισμού του επεξεργαστή προς σε άλλα σημεία του κώδικα.

Τέλος, η ιδιότητα της αναγκαστικής πρώτης εκτέλεσης του σώματος των εντολών της επαναληπτικής διαδικασίας, μπορεί να προκαλέσει εκτελεστικά λάθη (run-time-errors), τα οποία κατά κανόνα εκδηλώνονται διαστολικά, με διακοπή εκτέλεσης του προγράμματος, αλλά και στην χειρότερη περίπτωση με την εκτέλεση του προγράμματος, αλλά με σοβαρά λογικά λάθη που καταστρέφουν την ορθότητα της εξόδου από πλευράς αποτελεσμάτων.

4.5 ΟΙ ΕΝΤΟΛΕΣ WHILE ΚΑΙ DO-WHILE

Στις αντικειμενοστραφής γλώσσες προγραμματισμού εμφανίζεται η επαναληπτική διαδικασία με τον ακόλουθο τύπο:

```
While (υπόθεση)
{
Επικράτεια...εντολών;
}
```

όπου όσο η τιμή της υπόθεσης είναι true, θα εκτελείται και η επικράτεια εντολών του while, διαφορετικά η όλη επαναληπτική διαδικασία εκδηλώνει απόδραση προς την επόμενη εντολή του προγράμματος. Ένα πολύ απλό παράδειγμα είναι το ακόλουθο τμήμα κώδικα:

```
while (i <10)
 { Console.WriteLine("i=" + i);
   i--;
 }
```

τότε θα εμφανιστούν στην οθόνη οι τιμές του i όσο i<10. Η εντολή της επικράτειας ι -- είναι προγραμματιστικά εξαρτημένη με την υπόθεση του while και επομένως κάποτε θα σταματήσει η επαναληπτική διαδικασία.

Συχνά στον προγραμματισμό συμφέρει η έννοια της επαναληπτικής διαδικασίας όπου ο διακόπτης βρίσκεται μετά την επικράτεια των εντολών αντί

στην αρχή. Ο γενικός συντακτικός τύπος μιας τέτοιας εντολής είναι ο ακόλουθος:

```
Do
{
Επικράτεια...εντολών;
}
while(υπόθεση);
```

όπου αν η υπόθεση είναι ψευδής δημιουργείται **σημείο απόδρασης** στην επαναληπτική διαδικασία. Είναι φανερό πως η **επικράτεια της εντολής** θα εκτελεστεί τουλάχιστον μια φορά, ενώ φυσιολογικά θα πρέπει κάποια εντολή της επικράτειας να διαμορφώνει την τιμή της υπόθεσης της επαναληπτικής διαδικασίας.

ΠΡΟΤΑΣΗ

Η εντολή while είναι ισοδύναμη με την εντολή do-while

While (υπόθεση)
Επικράτεια εντολών;
	⇔	If(υπόθεση)
		Do
		Επικράτεια εντολών;
		While(υπόθεση);

όπου ο κίνδυνος της αναγκαστικής πρώτης εκτέλεσης της επικράτειας στο do-while, ελέγχεται με ένα if.

4.6 Η ΕΝΤΟΛΗ FOR

Οι αντικειμενοστρεφής γλώσσες προγραμματισμού διαθέτουν μια τουλάχιστον εντολή τύπου for, όπου κάποια **μεταβλητή έλεγχου (control variable)** παίρνει μια **αρχική τιμή (initialization)**, επιδρά σε μια επικράτεια εντολών και στην συνέχεια παίρνει μια επόμενη τιμή μέχρις ότου η μεταβλητή αυτή ξεπεράσει ένα ανώτερο φράγμα, όποτε και γίνεται απόδραση της επαναληπτικής διαδικασίας. Η εντολή for έχει γενικό συντακτικό τύπο:

```
for (αρχικοποίηση; Υπόθεση; ενημέρωση;)
{
επικράτεια εντολών;
}
```

Ένα παράδειγμα λειτουργίας του for είναι το ακόλουθο, όπου το πρόγραμμα διαβάζει τις τιμές της μεταβλητής I, μέσω της επαναληπτικής διαδικασίας και τις εμφανίζει στη οθόνη:

```
using System;
 namespace ConsoleApplication3
  {
    class Class1
     {
       static void Main(string[] args)
        { int i=0;
```

```
            for (i=1;i<10;i++)
            {
              Console.WriteLine(i);
            }
           }
      }
  }
```

Στις αντικειμενοστρεφείς γλώσσες προγραμματισμού για την εντολή for ισχύουν τα ακόλουθα:

- Οποιαδήποτε μεταβλητή ορισθεί εντός της εντολής for, η ορατότητα της δεν είναι δυνατή εκτός του for.
- Η υπόθεση του for πρέπει αναγκαστικά να είναι έκφραση ή και απλή μεταβλητή, τύπου Bool με τιμές true ή false.
- Είναι δυνατόν μέσα σε μια εντολή for, η αρχικοποίηση, η υπόθεση και η ενημέρωση να είναι κενές εντολές.

Αξιοσημείωτο είναι το φαινόμενο nesting με δυο ή και περισσότερα for, που η επικράτεια εντολών του εσωτερικού for πρέπει να ανήκει ολοκληρωτικά στην επικράτεια εντολών κάθε εξωτερικού for. Το αλυσιδωτό αυτό φαινόμενο είναι αναγκαίο σε περιπτώσεις που, για κάθε επανάληψη του εξωτερικού for, το εσωτερικό πρέπει να εξαντλήσει όλες τις συνδυαστικές του.

4.7 ΟΙ ΕΝΤΟΛΕΣ BREAK, GOTO

Συχνά μέσα στα πλαίσια μιας επαναληπτικής διαδικασίας ή και της ειδικής εντολής switch που θα αναλυθεί στην συνέχεια, υπάρχει **ανάγκη απόδρασης (escape position)**, δηλαδή πρόωρη διακοπή του προγράμματος στο αμέσως πιο εξωτερικό επίπεδο ή και σε κάποιο σημείο του προγράμματος που υπάρχει μια ορισμένη λογισμική παράγραφος. Στην JAVA/C# τέτοιου είδους ανάγκες εξυπηρετούνται με την εντολή break σε δυο συντακτικούς τύπους:

```
Break;
Break label;
```

Όπου το πρώτο χρησιμοποιείται για απόδραση από επαναληπτικές διαδικασίες και το switch ενώ στο δεύτερο έχουμε μεταφορά της λογικής ροής του προγράμματος εκτός κάποιας λογισμικής παραγράφου.

Στην περίπτωση που θέλουμε να μεταπηδήσουμε στο label γράφουμε μια από τις δύο εντολές και το όνομα του label. Στο επόμενο παράδειγμα σας δείχνουμε πως λειτουργεί :

```
class LabelTest
 {
    public static void main (String[] args)
   { int step;

theBlock:
```

```
        for(step=1;step<=10;step++)
         {  System.out.println("step is " + step );
            if (step > 6)
                break theBlock;
         }
         System.out.println("end of program");
      }
   }
```

Στη C#, η εντολή break δεν είναι εφαρμόσιμη με τον ίδιο τρόπο. Απλώς μεταπηδάμε έξω από μια επαναληπτική διαδικασία ή και το switch. Για να μεταπηδήσετε σε ένα block χρησιμοποιείστε τη goto εντολή.

Η εντολή της C# **goto** επιτρέπει ένα άλμα χωρίς συνθήκη σε μία εντολή ετικέτας (label). Η C# φιλοξένησε έξυπνα από τα πρώτα βήματα της την «αμαρτωλή» εντολή goto καλύπτοντας εύστοχα ένα μεγάλο κενό που άφησε η JAVA. Η εντολή goto μπορεί να εκτελέσει άλμα και προς τα μπροστά και προς τα πίσω μέσα στο κώδικα, αλλά το label πρέπει να είναι μέσα έκταση που μπορεί να καλύψει η εντολή. Επειδή η Java δεν εργάζεται στην πραγματικότητα με την goto εντολή, οι κοντινότερες σε αποτέλεσμα είναι break και continue, ωστόσο δεν παρουσιάζουν το ίδιο επίπεδο ευχρηστίας. Όταν χρησιμοποιούμε την εντολή goto πρέπει να προσέχουμε τα εξής:

- Χρησιμοποιώντας την goto για να βγούμε έξω από ένα try ή catch Βρόγχο θα έχει ως αποτέλεσμα να εκτελεστεί ο βρόγχος finally.
- Ένα λάθος του μεταγλωττιστή θα παρουσιαστεί εάν η goto χρησιμοποιηθεί για να βγει έξω από ένα βρόγχο finally

Στο παρακάτω παράδειγμα εφαρμόζουμε το προηγούμενο παράδειγμα αλλά σε C#.

```
using System;
    class LabelTest
     {
       static void Main(string[] args)
        { int step;
theBlock:
         for (step=1;step<=10;step++)
          {
            Console.WriteLine("step is " + step );
            if (step > 6)
               goto theBlock;
          }
          Console.WriteLine("end of program");
        }
       }
```

Παρατηρείται τις διαφορές στο συγκεκριμένα δύο προγράμματα. Με μια πρώτη ματιά εύκολα κατανοούμε τι θα εξάγει ο κώδικας και στις δύο περιπτώσεις. Το μεν πρόγραμμα της Java θα παράγει το εξής μήνυμα :

```
step is 1
step is 2
step is 3
step is 4
step is 5
step is 6
step is 7
end of program
```

Όμως παρόλο που ο καθένας θα υποψιαζόταν ότι το ίδιο θα παράγει και το πρόγραμμα της C# τα αποτελέσματα δεν είναι ίδια, γιατί θα τρέχει συνέχεια το κόμβο με τα step is 1 έως step is 7 και πάλι θα το επαναλάμβανε συνεχώς χωρίς να τερμάτιζε ποτέ το πρόγραμμα.

Προσοχή : Σε περιπτώσεις που το πρόγραμμα δε τερματίζει αλλά μπλέκει σε ατέρμονα loop όπως το παραπάνω πρόγραμμα της C# η μόνη λύση είναι να κρατάτε το κουμπί "Ctrl" και να πατάτε το κουμπί "$\frac{Pause}{Break}$" ή να κλείνετε το παράθυρο. Στη χειρότερη των περιπτώσεων πατήστε μαζί τα πλήκτρα "Alt", "Ctrl" και "Delete" και στο παράθυρο Task Manager πηγαίνετε στο tab "Applications" και πατήστε το κουμπί "End Task" έχοντας μαρκάρει το συγκεκριμένο πρόγραμμα.

Ο λόγος που το αποτέλεσμα στα δύο προγράμματα δεν είναι ίδιο είναι η διαφορά της λειτουργίας goto και break που προαναφέραμε.

4.8 Η ΕΝΤΟΛΗ CONTINUE

Η εντολή break προκαλεί **ολική απόδραση** από μια επαναληπτική διαδικασία. Ωστόσο στον προγραμματισμό υπάρχει συχνά η ανάγκη **μερικής απόδρασης** από μια επαναληπτική διαδικασία, δηλαδή την παράλειψη, της εκτέλεσης ενός υποσύνολου εντολών της επικράτειας για την παρούσα μόνο επανάληψη (Iteration). Τούτο επιτυγχάνεται με την εντολή **continue**.

4.9 Η ΕΝΤΟΛΗ SWITCH

Συχνά στα προγράμματα είναι χρήσιμο να υπάρχει ένας εύκολος τρόπος απόφασης για το ποιες εντολές θα εκτελεστούν σε σχέση με την τιμή κάποιας μεταβλητής ή έκφρασης που παίρνει τιμές που ορίζεται η ολική διάταξη (π.χ. ακέραιες) (**multi-way decision making**). Για παράδειγμα, στη γλώσσα Pascal υπάρχει η εντολή CASE-OF, ενώ άλλες γλώσσες αρκούνται στην χρήση μιας παρόμοιας εντολής (π.χ., Basic, Fortran, C++), όπου μάλιστα η μεταβλητή ή έκφραση ελέγχου μπορεί να πάρει και άλλες τιμές εκτός από ακέραιες.

Η C#, για να εκφράσει τέτοιες πολυπαραμετρικές αποφάσεις, διαθέτει την εντολή **switch**, την οποία πήρε από την C++, αλλά με ορισμένες διαφορές και ιδιαιτερότητες, με γενικό συντακτικό τύπο τον ακόλουθο:

```
switch (μεταβλητή ή έκφραση ελέγχου)
      {     case τιμή 1:
                  ομάδα εντολών 1;
            case τιμή 2:
                  ομάδα εντολών 2;
            ..............................
            case τιμή k:
                  ομάδα εντολών k;
            default:
                  ομάδα εντολών (k+1);
      }
```

όπου, η μεταβλητή ή έκφραση ελέγχου παίρνει τιμές τύπου **byte** ή **short** ή **int** ή **char**, ενώ οι τιμές **τιμήi** που σημειώνονται είναι ίδιου τύπου με τις τιμές της μεταβλητής ή έκφρασης ελέγχου και μάλιστα πρέπει να είναι **σταθερές** (literals) και όχι μεταβλητές. Αν η τιμή της μεταβλητής ελέγχου ταυτιστεί με κάποια **τιμήi**, τότε θα εκτελεστούν όλες οι **ομάδες εντολών w**, w=i, i+1, …, k, διαφορετικά, θα εκτελεσθεί μόνον η **ομάδα εντολών** (**k+1**) του **default**, το οποίο όμως δεν είναι αναγκαίο να υπάρχει μέσα σε ένα **switch**. Αν απουσιάζει και τύχει να μην υπάρχει **τιμήi** που να αντιστοιχεί στο switch, τότε απλούστατα δεν εκτελείται καμία εντολή και η ροή του προγράμματος μετατίθεται μετά το τέλος της επικράτειας της εντολής switch. Αξίζει επίσης να σημειωθεί, πως οι ομάδες εντολών μέσα στα case's, δεν είναι ανάγκη να συγκροτούν σύνθετη εντολή σε δομή λογισμικής παραγράφου (με τους χαρακτήρες { και }), οσεσδήποτε εντολές και αν περιέχουν, ούτε και τα case's έχουν καμία σχέση με τις ετικέτες (labels) εντολών που χρησιμοποιούν εντολές τύπου break.
Για παράδειγμα, η ακόλουθη παράγραφο, εκφράζει την εποχή που ανήκει ο μήνας month (ακέραιος):

```
int month = 3;
string season;
switch (month)
   { case 9:
     case 10:
     case 11:
       season = "Autumn";
       break;
     case 12:
     case 1:
     case 2:
       season = "Winter";
       break;
     case 3:
     case 4:
     case 5:
       season = "Spring";
       break;
     case 6:
     case 7:
     case 8:
       season = "Summer";
```

```
            break;
        default:
            season = "Unknown!";
        }
    Console.WriteLine("the month:"+month+" is in "+season);
```

Ο Αναγνώστης πρέπει να προσέξει ιδιαίτερα η ύπαρξη της εντολής break στο τέλος κάθε εποχής, διαφορετικά θα είχαμε το λανθασμένο αποτέλεσμα, ότι η αντίστοιχη εποχή είναι Summer, αντί για Spring που είναι το ορθό για την τιμή month το 3. Επίσης, σε ορισμένα case's δεν υπάρχει ούτε μία εντολή στην ομάδα τους, ούτε καν η κενή εντολή, γεγονός που επιτρέπει η σύνταξη της εντολής switch.

Είναι φανερό, πως στις περισσότερες περιπτώσεις, δεν είναι επιθυμητό να εκτελούνται όλες οι ομάδες εντολών από κάποιο case και μετά που συμπίπτει η τιμή του με εκείνη της μεταβλητής ελέγχου, αλλά αποκλειστικά και μόνο η ομάδα εντολών μόνον αυτού του case. Άλλωστε αυτό σημαίνει και πολλαπλός τρόπος απόφασης, γεγονός που τηρούν πιστά άλλες γλώσσες προγραμματισμού (π.χ. Pascal). Δυστυχώς, στην C++ και JAVA, αυτό μπορεί να γίνει κατασκευαστικά (με τρυκ), με την «έξυπνη» προσθήκη σημείου απόδρασης, με την βοήθεια της εντολής break, στο τέλος κάθε ομάδας εντολών, εκτός βέβαια της τελευταίας ομάδας που ανήκουν στο default. Σε μια τέτοια περίπτωση η εντολή switch ονομάζεται **switch/case-of** και ταυτίζεται με την εντολή CASE OF της Pascal. Ο συντακτικός τύπος, μιας τέτοιας εντολής switch παίρνει τελικά την ακόλουθη μορφή:

```
switch (μεταβλητή ή έκφραση ελέγχου)
    {   case τιμή 1:
            ομάδα εντολών 1;
            break;
        case τιμή 2:
            ομάδα εντολών 2;
            break;
        ……….
        case τιμή k:
            ομάδα εντολών k;
            break;
        default:
            ομάδα εντολών (k+1);
    }
```

4.10 Η ΕΝΤΟΛΗ FOREACH

Μια νέα εντολή της C# η **foreach** επαναλαμβάνεται διαμέσου των στοιχείων μιας συλλογής. Για τις προθέσεις της εντολής foreach, μια συγκέντρωση είναι :

- Κάθε τύπος που λειτουργεί στη διασύνδεση του System.IEnumerable.
- Κάθε πίνακας System.Array λειτουργεί με IEnumerable.
- Κάθε τύπος που λειτουργεί ισάξια με το IEnumerable.

Η σύνταξη της εντολής foreach είναι :

foreach (τύπος προσδιοριστικό in έκφραση)
{επικράτεια εντολών; }

Το επόμενο παράδειγμα της εντολής foreach επαναλαμβάνεται μέσα από στοιχεία string :

```
string[] days = new string[] {"Monday", "Tuesday", "Wednesday",
   "Thursday", "Friday", "Saturday", "Sunday"};
          foreach (string theDays in days)
           {
             System.Console.WriteLine(theDays);
           }
```

και το αποτέλεσμα είναι να εκτυπώνει τις ημέρες που έχουμε στη συγκέντρωση από αλφαριθμητικά στο string days.

Αυτό που πρέπει να προσέξουμε στη χρήση της εντολής foreach είναι :

- Ο τύπος της επαναλαμβανόμενης μεταβλητής (για το παράδειγμα theDays) πρέπει να είναι του ίδιου τύπου με αυτή της συγκέντρωσης των στοιχείων (για το παράδειγμα days).
- Ο μεταγλωττιστής παράγει σφάλμα εάν γίνει μια απόπειρα να οριστεί μια τιμή στην μόνο για ανάγνωση επαναλαμβανόμενη μεταβλητή μέσα στην εντολή foreach.

4.11 ΑΛΛΕΣ ΕΝΤΟΛΕΣ ΤΗΣ C#

Στην παράγραφο αυτή, θα μελετήσουμε άλλες εντολές της C#. Θα γίνει μια απλή αναφορά των εντολών αυτών, αφού οι περισσότερες από αυτές θα αναλυθούν σε επόμενα κεφάλαια. Οι εντολές αυτές είναι:

Η εντολή return, οι εντολές εξαιρέσεων try, catch, finally, throw. Οι εντολές check και uncheck η εντολή συγχρονισμού νημάτων lock και τέλος, η εντολή κλήσης πακέτων using.

4.12 ΑΣΚΗΣΕΙΣ

1. Να γραφεί πρόγραμμα C#, όπου ο χρήστης δίνει στη γραμμή εκτέλεσης (command line) τις τιμές των παραμέτρων a,b,c και e και δίνονται στην έξοδο οι ρίζες του τριώνυμου ax^2+bx+c με προσέγγιση e (π.χ. 0.0001).Παρακάτω δίνεται ο κώδικας του προγράμματος σε JAVA.

```
import java.math.*;

public class Trionym
{
  public static void main(String args[ ])
   { double a,b,c,d,r1,r2,e;
     a=(new Double(args[0])).doubleValue();
```

```
b=(new Double(args[1])).doubleValue();
c=(new Double(args[2])).doubleValue();
e=(new Double(args[3])).doubleValue();
if((Math.abs(a)<e)&&(Math.abs(b)<e))
 { System.out.println("It's a trivial case…");
 }
 else if(Math.abs(a)<e)
 { System.out.println("One simple solution="+(-(c/b)));
 }
 else if(Math.abs (c)<e)
 { System.out.println("First solution=0, second"+(-(b/a)));
 }
 else
 { r1=-b/(2*a);
   d=b*b-4*a*c;
   r2=Math.sqrt(Math.abs(d))/(2*a);
 if((d>0.0)||(Math.abs(d)<e))
  { System.out.println("The solutions are:"
    +(r1+r2)+","+(r1-r2));
  }
 else
  { System.out.println("The non real solutions" + "are:"
    +r1+ "+i" +r2 +"," +r1+"-i "+r2);
  }
 }
}
```

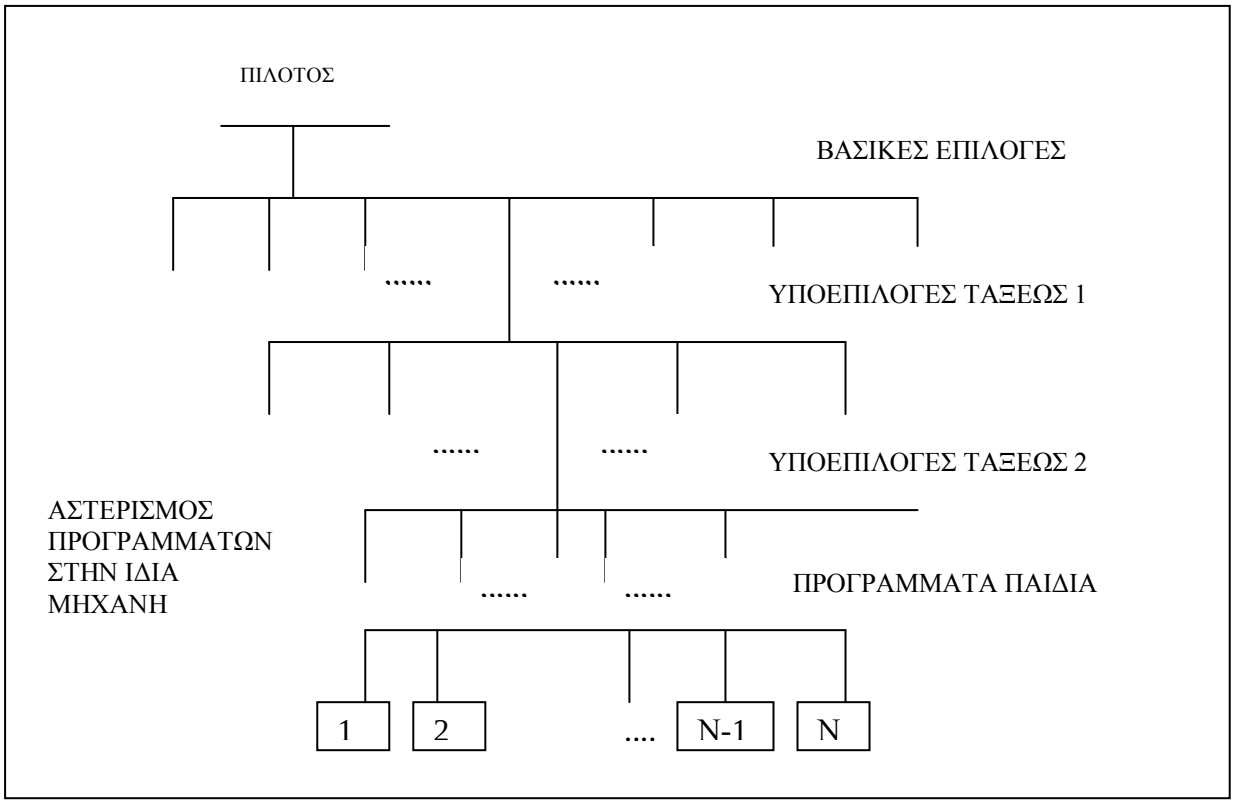
2. Να γραφεί πρόγραμμα που θα υπολογίζει το άθροισμα $\Sigma^{\kappa}_{\iota=1}\ \alpha_{\iota}$ =sum;

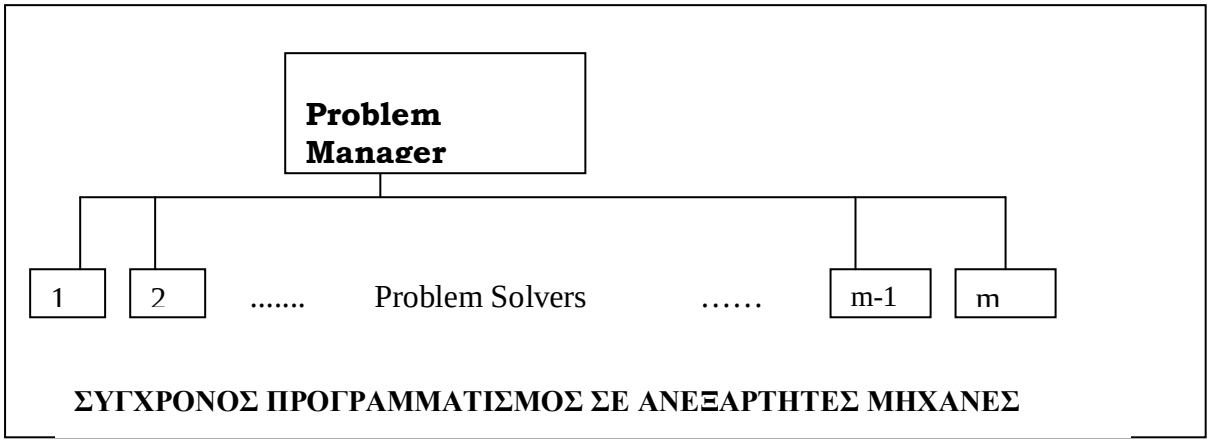
ΚΕΦΑΛΑΙΟ 5^{Ο}

5. ΝΗΜΑΤΑ ΚΑΙ ΣΥΓΧΡΟΝΙΣΜΟΣ

5.1 ΠΟΛΥΔΙΕΡΓΑΣΙΑ

Η έννοια του παράλληλου ή/και σύγχρονου προγραμματισμού (**Concurrent Programming**) έγιναν γνωστές στον προγραμματισμό από τα πρώτα Λειτουργικά Συστήματα των mainframes της δεκαετίας του '70 που υποστήριζαν δυναμικά πολυπρογραμματισμό και ειδικότερα πολυδιεργασία (**Multitasking**). Κατά κανόνα, ένα αρχικό πρόγραμμα (πρόγραμμα πατέρας) (**Parent process**) μπορεί να θέτει σε εκτέλεση άλλες διαδικασίες (προγράμματα παιδιά) (child process) και όλα αυτά μαζί να εκτελούνται ταυτόχρονα μέχρι το τέλος της όλης εφαρμογής. Μια τέτοια κατάσταση ονομάζεται σήμερα **Αστερισμός προγραμμάτων** και συχνά στην πράξη, υλοποιείται με ένα **πρόγραμμα πιλότο** (πρόγραμμα πατέρας), ο οποίος διαχειρίζεται βασικές και δευτερεύουσες επιλογές που ο χρήστης γραφικά επιλέγει και ανάλογα εκτελούνται αντίστοιχα προγράμματα παιδιά. Υπάρχει, όμως και η περίπτωση που μια μηχανή σε δίκτυο (ή/και διαδίκτυο) εκτελεί χρέη **Problem manager,** η οποία συντονίζει τα αντίστοιχα προγράμματα άλλων μηχανών (**Problem Solvers**), με την έννοια ότι ένα δύσκολο πρόβλημα συνήθως εκθετικού υπολογιστικού χρόνου αναλύεται σε μια ακολουθία υποπροβλημάτων, τα οποία δίνονται σε επιλυτές κάτω από αλγοριθμικό συντονισμό.

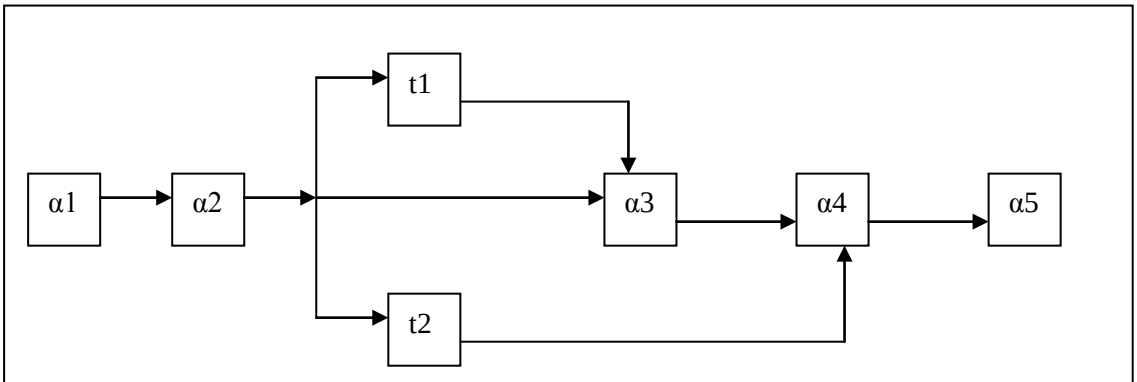




Ωστόσο, το multitasking από τη δεκαετία του '90 και μετά, διακρίνεται σε δυο μεγάλες, διαφορετικές μεταξύ τους κατηγορίες:

Multitasking-Process- Based (MPB): είναι στην ουσία η δυνατότητα των μηχανών να εκτελούν δυο και περισσότερα προγράμματα σε συγχρονισμένη βάση. Πρόκειται για εκτελέσιμες μονάδες (Units) (πχ *.exe, *.com, *.bat) οι οποίες ανταλλάσσουν μεταξύ τους μηνύματα, δεδομένα, αρχεία, με κύριο στόχο την αριστοποίηση του αναμενόμενου υπολογιστικού χρόνου της όλης εφαρμογής.

Multitasking- Thread-Based (MTB): Είναι η μυωπική αντιμετώπιση του multitasking, όπου τα **Νήματα** (threads) αποτελούν τμήματα κώδικα τα οποία αποτελούνται από μια ακολουθία εντολών, τα οποία όμως δεν μπορούν να τρέξουν μόνα τους αλλά μόνο μέσα από το πρόγραμμα στο οποίο και ανήκουν. Ο Διαδικτυακός προγραμματισμός διαθέτει παράλληλα νήματα, όπου το πρόγραμμα εκμεταλλεύεται το υπολογιστικό περιβάλλον, χρησιμοποιώντας τη στοίβα εκτέλεσης και μετρητή εντολών του περιβάλλοντος. Η C++, η Java και η C#, λειτουργούν αποτελεσματικά με MTB με βασικό στόχο την ελαχιστοποίηση του αναμενόμενου υπολογιστικού χρόνου του ίδιου του προγράμματος.



MTB: Παράδειγμα Κώδικα Προγράμματος με δύο Νήματα παράλληλα t1 και t2

Κάθε νήμα εκτέλεσης είναι μια ανεξάρτητη ροή του ελέγχου εκτέλεσης, μέσα στο πρόγραμμα. Στο παραπάνω σχήμα δίνεται ένα παράδειγμα προγράμματος όπου ο κώδικας χωρίζεται σε περιοχές α1, α2, ..., α5. μετά την περιοχή α2 αρχίζουν τα παράλληλα νήματα t1,t2. το πρώτο θα περατωθεί από την περιοχή α3 και το δεύτερο στην α4. αν δεν υπήρχαν τα νήματα t1,t2, τότε ο αναμενόμενος υπολογιστικός χρόνος θα ήταν μεγαλύτερος, διότι το πρόγραμμα θα έπρεπε να εκτελεστεί **σειριακά** σε κάθε περιοχή του, όπως για παράδειγμα: α1, α2, t1, t2, α3, α4, α5. προσοχή, είναι φανερό ότι ισχύει η σχέση:

Αναμενόμενος υπολογιστικός χρόνος (t1)+ Αναμενόμενος υπολογιστικός χρόνος (t2) >= Αναμενόμενος υπολογιστικός χρόνος (t1+t2)

Δηλαδή, ο αναμενόμενος υπολογιστικός χρόνος των παράλληλων περιοχών κώδικα (t1+t2) είναι πάντοτε μικρότερος από το άθροισμα των αντίστοιχων χρόνων των t1 και t2.

5.2 ΒΑΣΙΚΕΣ ΙΔΙΟΤΗΤΕΣ ΝΗΜΑΤΩΝ

Όταν αρχίζει η εκτέλεση ενός προγράμματος Java ή C#, στην ουσία αρχίζει ένα **βασικό νήμα εκτέλεσης.** Μάλιστα, μπορεί να υπάρχουν και άλλα νήματα τα οποία δεν είναι εμφανή στο χρήστη. Για παράδειγμα, στην περίπτωση αυτοτελών, ανεξάρτητων εφαρμογών Java ή C#, έκτός από το βασικό νήμα, τρέχει παράλληλα και ο συλλέκτης σκουπιδιών.

Στην Java ειδικότερα, ένα νήμα εκτέλεσης μπορεί να βρίσκεται, κάθε στιγμή, σε μια από τις ακόλουθες καταστάσεις:

- **Ready to Run:** Όταν το νήμα πάρει CPU-time μετά την αρχικοποίησή του με κατάλληλο αντικείμενο με τη βοήθεια της δήλωσης new.
- **Running:** Ένα νήμα που βρίσκεται στην κατάσταση ready to run μπορεί να τρέξει, αν κληθεί η ειδική μέθοδος **start** η οποία λειτουργεί με τη μέθοδο **run** του νήματος
- **Waiting:** Ένα νήμα σε κατάσταση running μπορεί να σταματήσει προσωρινά την εκτέλεσή του με ειδικές κλήσεις σε μεθόδους
- **Done:** Όταν τελειώσει φυσιολογικά η μέθοδος run του νήματος στην κατάσταση running, τότε τελειώνει η εκτέλεση του νήματος.

Το .NET έχει μια προκαθορισμένη ομάδα από στάδια τα οποία οριοθετούν των κύκλο ζωής τους. Αυτά τα στάδια ορίζονται μέσα στο **System.Threading.ThreadState**. Από την άλλη μεριά η κατάσταση στην οποία βρίσκεται ένα νήμα στη C# φαίνεται από τον παρακάτω πίνακα.

Πίνακας: Το System.Threading.ThreadState

Στάδια νήματος	Περιγραφή
Aborted	Το νήμα είναι στο στάδιο **Stopped**
AbortRequested	Η μέθοδος **Abort** έχει καλεστεί για το νήμα, αλλά το **ThreadAbortException** δεν έχει ακόμα επιλεχθεί.
Background	Το νήμα εκτελείται σαν ένα νήμα δευτερεύουσας προτεραιότητας. Αυτό είναι παρόμοιο με το νήμα **daemon της Java**.
Running	Το νήμα ξεκίνησε. Δε μπορεί να μπλοκαριστεί και καμία

	κλήση δεν του προκαλεί Abort.
Stopped	Το νήμα σταματάει.
StopRequested	Είναι για εσωτερική χρήση μονό.
Suspended	Το νήμα θέτεται σε διαθεσιμότητα μέσω της μεθόδου **Suspend**.
SuspendRequested	Το νήμα επιλέγεται να τεθεί σε διαθεσιμότητα.
Unstarted	Το νήμα δημιουργείται αλλά δεν ξεκινάει μέσω της μεθόδου **Start**.
WaitSleepJoin	Το νήμα μπλοκάρετε επειδή μιας κλήσης από τις ακόλουθες μεθόδους : **Wait, Sleep ή Join.**

Αυτά τα στάδια είναι διαθέσιμα μόνο για ανάγνωση (read-only) ιδιότητες **Thread.ThreadState**. Τα νήματα μπορούν να είναι σε παραπάνω από ένα στάδια, άρα είναι σημαντικό να ελέγχουμε τα στάδια με προσοχή. Για παράδειγμα, εάν ένα νήμα είναι μπλοκαρισμένο για το λόγο μιας κλήσης στην κλάση **Monitor.Wait** κάποιο άλλο νήμα καλεί την **Abort**, τότε το μπλοκαρισμένο νήμα θα είναι και αυτό στο στάδιο **WaitSleepJoin** και **AbortRequested** Όταν το μπλοκαρισμένο νήμα επιστρέψει από την Wait κλήση ή διακοπεί, το στάδιο θα είναι **AbortRequested** και η **ThreadAbortException** θα ενεργοποιηθεί, οδηγώντας σε ένα επίπεδο Aborted.

Αξίζει να επισημανθούν τα ακόλουθα :

- Η ιδιότητα **IsAlive** επιστρέφει true εάν στο στάδιο νήματος δεν περικλείονται τα **Unstarted, Stopped** και **Aborted.**

- Η ιδιότητα **IsBackground** επιστρέφει true εάν το νήμα τρέχει με δευτερεύουσα προτεραιότητα . Η μέθοδος **isDaemon** της Java επιφέρει παρόμοια αποτελέσματα. Νήματα με δευτερεύουσα προτεραιότητα (background threads) είναι ίδια με τα νήματα με **Normal προτεραιότητα** εκτός από ότι δεν αποτρέπουν μια διαδικασία από το τερματισμό. Εάν η διαδικασία δεν έχει ενεργό νήμα πρωτεύον προτεραιότητας (active foreground thread), η κλήση **Abort** θα επιλεχθεί σε κάθε τρέχον νήμα δευτερεύουσας προτεραιότητας και η διαδικασία θα τερματιστεί.

Στο παρακάτω πίνακα υπάρχει λίστα ενεργειών και τα αποτελέσματα που έχουν πάνω στα στάδια νήματος.

Πίνακας: Αποτελέσματα ενεργειών πάνω στα στάδια νημάτων

Εκπλήρωση ενέργειας	**Το στάδιο νήματος γίνεται...**
Ένα νέο νήμα αρχίζει	**Unstarted**
Ένα νήμα καλεί το **Start**	**Running**
Το νήμα αρχίζει να εκτελείται	**Running**
Το νήμα καλεί το **Sleep**	**WaitSleepJoin**
Το νήμα καλεί το **Wait** πάνω σε άλλο **Object**	**WaitSleepJoin**
Άλλο νήμα καλεί το **Interrupt**	**Running**
Άλλο νήμα καλεί το **Suspend**	**SuspendRequested**
Το νήμα θέτεται σε διαθεσιμότητα	**Suspended**
Άλλο νήμα καλεί το **Resume**	**Running**
Άλλο νήμα καλεί το **Abort**	**AbortRequested**

Το νήμα ανταποκρίνεται σε μια αίτηση για αποτυχία (abort)	**Stopped**
Το νήμα τερματίζει	**Stopped**

Όταν χρησιμοποιούμε τα στάδια των νημάτων, έχετε στο νου σας ότι μπορούν να αλλάξουν γρήγορα και να ανατρέπουν τη κλήση που επρόκειτο να κάνουν. Επειδή το νήμα έχει ένα στάδιο του **Suspended** όταν ελέγχεται δεν είναι αναγκαίο να σημαίνει ότι θα παραμείνει για πολύ ώρα μέχρι ένα άλλο νήμα καλέσει τη **Resume**. Για το λόγο αυτό είναι σημαντικό να ελέγχουμε για πιθανή εξαίρεση όταν χρησιμοποιούμε κλήσεις νημάτων

Ένα νήμα εκτέλεσης, χαρακτηρίζεται από το όνομά του, την προτεραιότητα εκτέλεσής του, και το όνομα της ομάδας νημάτων που ανήκει, ακολουθώντας το γενικό συντακτικό τύπο:

Thread [νήμα, προτεραιότητα, ομάδα]

Σε πολλές περιπτώσεις, υπάρχει πρόβλημα **συγχρονισμού** των παράλληλων νημάτων εκτέλεσης, ιδιαίτερα όταν διαχειρίζονται κοινούς πόρους, όπου η πρόσβαση σε αυτούς πρέπει να ακολουθεί ορισμένους κανόνες.

Για να οριστεί ένα νήμα εκτέλεσης θα πρέπει να δημιουργηθεί ένα αντικείμενο πάνω στην παρεχόμενη κλάση, **System.Threading.** Μια εφαρμογή για τα όσα είπαμε βλέπουμε στη συνέχεια. Το ίδιο πρόγραμμα είναι γραμμένο πρώτο σε Java και στη συνέχεια σε C#.

```
Public class NewThread implements Runnable
{ public void run ()
  { for (int i=0;i<10;i++)
    { System.out.println("Counter:" +i);
    }
  }
  public new thread() throws Exception
   { Thread exthread=new Thread(this);
     exthread.start();
   }
}
```

Στο παραπάνω παράδειγμα, η κλάση NewThread της Java χρησιμοποιεί το Runnable interface το οποίο περνά σαν παράμετρο στο νέο instance του Thread. Για τη C# το ίδιο πρόγραμμα γίνεται:

```
using System;
using System.Threading;

namespace NewThread
 {
  class NewThread
   {
    public void run ()
  {
```

```
    for (int i=0;i<10;i++)
     { Console.WriteLine.("Counter:" +i);
     }
   }
   public new thread ()
    {
     Thread exthread=new Thread(new ThreadStart(run));
     exthread.Start();
    }
   }
  }
```

Εδώ θα πρέπει να επισημάνουμε τα εξής:

- Η **System.Threading** εισάγεται πάντα μέσα σε προγράμματα που χρησιμοποιούν threads.
- Το **ThreadStart** δέχεται οποιαδήποτε μέθοδο που δεν έχει ορίσματα και επιστρέφει τύπο **void.**
- Η μέθοδος που είναι περασμένη στο ThreadStart δεν είναι ανάγκη να καλείται run και να είναι public.
- Το **.NET** δεν έχει μια ισοδύναμη διαδικασία σαν της **ThreadGroup** της Java.

5.3 ΝΗΜΑΤΑ ΜΕ ΒΑΣΗ ΤΗ ΔΙΑΠΡΟΣΩΠΕΙΑ ThreadStart

Στη Java ένας τρόπος δημιουργίας νημάτων εκτέλεσης, μπορεί να γίνει με τη δημιουργία μιας νέας κλάσης για τη νέα εργασία που να υλοποιεί τη **διαπροσωπεία Runnable.** Από την άλλη μεριά η C# για τη ίδια λειτουργία χρησιμοποιεί την μέθοδο **ThreadStart.** Η ThreadStart, δηλώνει μόνο μία μέθοδο, την **run**, η οποία βέβαια εκτελείται αυτόματα από τη μέθοδο **Start.** Εδώ η μέθοδος run μοιάζει με τη μέθοδο main του βασικού νήματος, **όπου υπάρχει ο κώδικας όλων των νημάτων που θα εκτελεστούν,** τα οποία θα τερματιστούν όταν περατωθεί και η εκτέλεση μέσα στη μέθοδο αυτή. Μπορεί να χρησιμοποιεί άλλες κλάσεις, μεθόδους και μεταβλητές όπως και η μέθοδος main.

Μετά τη δημιουργία της νέας κλάσης που θα υλοποιεί την ThreadStart, θα πρέπει να οριστεί ένα αντικείμενο τύπου Thread, το οποίο **θα αρχικοποιηθεί μέσα στην κατασκευαστική μέθοδο της κλάσης αυτής.** Η αρχικοποίηση θα πρέπει να γίνει με έναν από τους constructors της κλάσης Thread με γενικό τύπο:

Thread(ThreadStart, String name)

Όπου **name** είναι το όνομα του νήματος που θα αρχίσει την εκτέλεσή του, αφού δοθεί βέβαια η μέθοδος **start**. Όλα αυτά θα πρέπει να δοθούν μέσα στον κατασκευαστή της κλάσης που υλοποιεί την ThreadStart ακόμη και η μέθοδος start. Η **Main** μέθοδος το μόνο που απαιτείται να έχει είναι ο ορισμός κάθε νήματος με κάποιο αντικείμενο της κλάσης που υλοποιεί την ThreadStart, κατά κανόνα περνώντας και το όνομα του νήματος μέσα από την κατασκευαστική μέθοδο της κλάσης αυτής.

Συνοψίζοντας τα παραπάνω, παρουσιάζουμε ένα πίνακα με τα βασικά νηματικά μελών της Java και C#.

Πίνακας: Σύγκριση νηματικών μελών της Java και C#

Java	C#	Σχόλια
Runnable	ThreadStart	Εντολές (delegate)
Thread.currentThread	Thread.CurrentThread	Ιδιότητα (property)
Thread.getName	Thread.Name	Ιδιότητα
Thread.getPriority	Thread.Priority	Ιδιότητα. Το .NET χρησιμοποιεί ένα αριθμητικό για να ορίζει τα στάδια των νημάτων.
Thread.setPriority	Thread.Priority	Ιδιότητα. Το .NET χρησιμοποιεί ένα αριθμητικό για να ορίζει τα στάδια των νημάτων.
Thread.getThreadGroup	N/A	Ιδιότητα
Thread.interrupt	Thread.Interrupt	Μέθοδος (method)
Thread.interrupted	N/A	Μέθοδος
Thread.isInterrupted	N/A	Μέθοδος
Thread.isDaemon	Thread.isBackGround	Ιδιότητα
N/A	Thread.isAlive	Ιδιότητα
N/A	Thread.isThreadPoolThread	Ιδιότητα
Thread.join	Thread.Join	Μέθοδος
Thread.suspend(dep)	Thread.Suspend	Μέθοδος
Thread.resume(dep)	Thread.Resume	Μέθοδος
Thread.run	N/A	Το .NET χρησιμοποιεί το ThreadStart για να ορίσει νήματα.
Thread.sleep	Thread.Sleep	Μέθοδος
Thread.start	Thread.Start	Μέθοδος
Thread.stop	Thread.Abort	Μέθοδος
N/A	Thread.ResetAbort	Μέθοδος
N/A	Thread.SpinWait	Μέθοδος
Thread.yield	Thread.Sleep(0)	Μέθοδος

5.4 ΣΥΓΧΡΟΝΙΣΜΟΣ

Είναι αρκετές οι περιπτώσεις, που είναι ανάγκη, δυο ή και περισσότερα νήματα να διαχειριστούν μια παράγραφο του προγράμματος ή μια μέθοδο ταυτόχρονα. Εδώ είναι επιθυμητό να υπάρχει συγχρονισμός στις εργασίες που επιτελούνται, διαφορετικά τα αποτελέσματα ενδέχεται να οδηγήσουν να οδηγήσουν σε **λογισμικές καταστροφές.** Αν για παράδειγμα, ένα νήμα διαβάζει μια μεταβλητή την ίδια στιγμή που ένα άλλο νήμα εκχωρεί τιμή στην μεταβλητή αυτή το αποτέλεσμα είναι απρόβλεπτο.

Γλώσσες προγραμματισμού αντικειμενοστραφούς τεχνολογίας προσπαθούν να πετύχουν **συγχρονισμό (synchronization).** Ειδικά η Java έχει ένα εύχρηστο μηχανισμό γα το συγχρονισμό των νημάτων εκτέλεσης, που

χρησιμοποιούν ένα ή περισσότερα κοινά μέσα. Ακολουθεί τον τρόπο συγχρονισμού της τεχνικής του **monitor** ή/και **semaphore**. Το monitor είναι ένα αντικείμενο που είναι **κλειδωμένο (lock)** επειδή χρησιμοποιείται από κάποιο νήμα αποκλειστικά, ή είναι διαθέσιμο σε απαιτήσεις. Ο συγχρονισμός στη Java γίνεται με τη δήλωση της μεθόδου που ακολουθεί τον συντακτικό τύπο ο οποίος φαίνεται στο παρακάτω παράδειγμα:

```
Public void dosomething ()
 {
   synchronized(this)
    {
     //some synchronized operation
    }
 }
```

Η C# ακολουθεί την ίδια δομή με τη Java και την ίδια λογική με μόνη αλλαγή τη χρήση του keyword **lock** αντί για το keyword synchronized. Έτσι, η δομή του synchronization στη C# φαίνεται στο παρακάτω παράδειγμα.

```
Public void dosomething ()
 {
  lock(this)
   {
    //some synchronized operation
   }
 }
```

Η C# δεν υποστηρίζει μια ευθέως αναλογία για τη χρήση της Java keyword synchronized σαν προσαρμογέας μεθόδου, αλλά τα ίδια αποτελέσματα μπορούν να επεξηγήσουν τη μέθοδο μαζί με το **MethodImpl (MethodImplOptions.Synchronized)** χαρακτηριστικό. Οπότε μια συγχρονισμένη μέθοδος στη Java :

```
private synchronized void doLock()
{
  // εκτέλεσε το κώδικα μέσα στο μπλΟκ
}
```

μοιάζει με το ακόλουθο παράδειγμα της C# :

```
[MethodImpl(MethodImplOptions.Synchonized)]
private void doLock() {
          // εκτέλεσε το κώδικα μέσα στο μπλΟκ
 }
```

Το χαρακτηριστικό **MethodImpl** ορίζεται μέσα στο χώρο ονομάτων (namespace) **System.Runtime.CompilerServices**, έτσι αυτό πρέπει να εισαχθεί πριν η μέθοδος εκτελεστεί. Τα σημεία κλειδιά που πρέπει να αναφέρουμε είναι τα παρακάτω:

Περιορισμένη αποθήκευση εκτελείται πάνω από το αντικείμενο **lock**, έτσι τύποι τιμών όπως ακέραιοι ή αριθμοί κινητής υποδιαστολής δεν μπορούν

να χρησιμοποιηθούν για να εκπληρώνουν κλειδώματα. Εάν αποπειραθείτε να χρησιμοποιήσετε τύπους τιμών το αποτέλεσμα που θα έχετε είναι σφάλμα κατά την μεταγλώττιση.

Για να προστατέψετε μια μέθοδο static ή μεταβλητή, η χρήση του this δεν είναι αναγκαία αλλά είναι πιθανή η χρήση της επιστροφής από **typeof(MyClass).**

5.4.1 ΣΥΓΧΡΟΝΙΣΜΟΣ ΑΝΕΞΑΡΤΗΤΩΝ ΝΗΜΑΤΩΝ

Στην Java, η κλάση **Object** περιέχει τις μεθόδους **wait, notify, notifyAll**. Η κλάση **Object** της C# δεν περικλείει ισοδύναμες μεθόδους, αλλά η κλάση **System.Threading.Monitor** καλύπτει τους προγραμματιστές με την απαιτούμενη λειτουργικότητά της στο τομέα αυτό, όπως φαίνεται στο παρακάτω πίνακα.

Πίνακας:Η κλάσης Object της Java και μεθόδων Threading της C#

Java	C#
Object.wait()	Monitor.Wait(αντικείμενο)
Object.notify()	Monitor.Pulse(αντικείμενο)
Object.notifyAll()	Monitor.PulseAll(αντικείμενο)

Η μόνη σημαντική διαφορά είναι ότι η κλάση **Monitor** απαιτεί ο προγραμματιστής να επιλέξει ένα αντικείμενο πάνω στο οποίο θα γίνει το κλείδωμα. Όπως στην Java, η C# απαιτεί αυτές τις κλήσεις να εκπληρώνονται μέσα σε ένα συγχρονισμένο τομέα του κώδικα, ο οποίος είναι συμπληρωμένος χρησιμοποιώντας το **keyword lock.**

Η κλάση **Monitor** επίσης παρέχει περισσότερη λειτουργικότητα από ότι η Java, δίνοντας στον προγραμματιστή περισσότερο έλεγχο διαχείρισης για το πώς τα κλειδώματα πετυχαίνονται και αποδεσμεύονται.

Όλα τα παραπάνω είναι εμφανή στο παρακάτω πρόγραμμα Java.

```
// ΣΥΓΧΡΟΝΙΣΜΕΝΑ ΑΝΕΞΑΡΤΗΤΑ ΝΗΜΑΤΑ ΧΩΡΙΣ ΕΣΩΤΕΡΙΚΗ ΕΠΙΚΟΙΝΩΝΙΑ
class Cshare
 { synchronized void display1(int numb, String threadname)
   { System.out.println("i="+numb+" "+threadname);
   }
   synchronized void display2(int numb, String threadname)
    { System.out.println("i="+numb+" "+threadname);
    }
  }
   class NewThread implements Runnable
   { String name;
     Cshare object;
     public NewThread1(Cshare obj,String strname)
     {
     object=obj;
     name=strname;
```

```
      System.out.println("Starting:"+name);
     }
     public void run()
      { for (int i=1;i<6;i++)
        { object.display1(i,name);
        }
        System.out.println("ok ➔"+name);
      }
    }
class NewThread2 implements Runnable
 {
   String name;
   Cshare object;
   public NewThread2(Cshare obj,String strname)
    {
     object=obj;
     name=strname;
     System.out.println("Starting:"+name);
    }
    public void run()
     {
      for (int i=1;i<6;i++)
      {
        object.display2(i,name);
      }
      System.out.println("ok ➔"+name);
      }
   }
class Java041
 { public static void main(String x[])
    { Cshare common = new Cshare();
       new Thread(new NewThread1( common,"Given")).start();
       new Thread(new NewThread2( common,"Taken")).start();
    }
 }
```

Το παραπάνω πρόγραμμα γράφεται ισοδύναμα σε C# ως ακολούθως:

```
using System;
class Cshare
{ internal virtual void  display1(int numb, System.String
    threadname)
  { lock(this)
    { System.Console.WriteLine("i =" + numb +" " + threadname);
    }
  }
 internal virtual void  display2(int numb, System.String
    threadname)
  { lock(this)
```

```
     { System.Console.WriteLine("i =" + numb +" " +
       threadname);
     }
  }
}
class NewThread1 : SupportClass.IthreadRunnable
 {  internal System.String name;
    internal Cshare object_Renamed;
    public NewThread1(Cshare obj, System.String strname)
     { object_Renamed = obj;
       name = strname;
      System.Console.WriteLine("Starting:"+ name);
     }
     public void  Run()
      { for (int i = 1; i < 6; i++)
         { object_Renamed.display1(i, name);
         }
      System.Console.WriteLine("ok==>" + name);
      }
  }
class NewThread2 : SupportClass.IthreadRunnable
{ internal System.String name;
  internal Cshare object_Renamed;
  public  NewThread2(Cshare obj, System.String strname)
   { object_Renamed = obj;
      name = strname;
      System.Console.WriteLine("Starting:"+ name);
   }
    public void  Run()
     { for (int i = 1; i < 6; i++)
        {  object_Renamed.display2(i, name);
        }
       System.Console.WriteLine("ok==>" + name);
     }
   }
class Java041
{   [STAThread]
   public static void  Main(System.String[] x)
    { Cshare common = new Cshare();
      new System.Threading.Thread(new
      System.Threading.ThreadStart(new NewThread1(common,
      "Given").Run)).Start();
      new System.Threading.Thread(new
      System.Threading.ThreadStart(new NewThread2(common,
     "Taken").Run)).Start();
    }
}
```

Δυστυχώς, παρόλο που οι μέθοδοι display1, display2 είναι συγχρονισμένες, η έξοδος εκτέλεσης του προγράμματος οδηγεί σε λογική ανακολουθία:

```
Starting: Given
Starting: Taken
i=1 Taken
i=2 Taken
i=3 Taken
i=4 Taken
i=1 Given
i=5 Taken
ok ➔ Taken
i=2 Given
i=3 Given
i=4 Given
i=5 Given
ok ➔ Given
```

5.5 ΑΣΚΗΣΕΙΣ

1. Να μετατραπεί ο παρακάτω κώδικας Java σε ισοδύναμο κώδικα C#.

```
class NewTread extends Thread
 {
   NewThread(String name)
   { super(name);
   }
   public void run()
   { for (int i=1;i<6;i++)
      { System.out.println("i="+ i+""+getName());
        try
         { sleep((int)(Math.random()*100));
         }
        catch(InteruptedException e){}
       }
       System.out.println("Ok➔"+ getName());
    }
   }
class Java033
 { public static void main(String x[])
   { new NewThread("winner").start();
     new NewThread("looser").start();
    }
  }
```

2. Να μετατραπεί ο παρακάτω κώδικας Java σε ισοδύναμο κώδικα C#.

```
class Cshare
 {
```

```
   boolean flag = true;
   synchronized void display1(int numb, String threadname)
    {
     if (!flag)
       try
        { wait();
        }
        catch(InterruptedException e){}
        System.out.println("i=" +numb + ""+threadname);
        flag =false ;
        notify();
    }
synchronized void display2(int numb, String threadname)
 { if (!flag)
    try
     { wait();
     }
     catch(InterruptedException e){}
     System.out.println("i=" +numb + ""+threadname);
     flag =true ;
     notify();
 }
}
class NewThread1 implements Runnable
 { String name;
   Cshare object;
    public NewThread1(Cshare obj, String strname)
     { object=obj;
       name=strname;
       System.out.println("Starting: "+name);
     }
   public void run()
    { for (int i=1;i<6; i++)
      object.display1(i,name);
     System.out.println("ok➔" +name);
    }
  }
class NewThread2 implements Runnable
 { String name;
   Cshare object;
   public NewThread2(Cshare obj, String strname)
    { object=obj;
      name=strname;
      System.out.println("Starting: "+name);
    }
    public void run()
 { for (int i=1;i<6; i++)
  object.display2(i,name);
```

```
  System.out.println("ok➔" +name);
 }
}
class Java040
 { public static void main(String x[])
    { Cshare common=new Cshare();
      new Thread(new NewThread1(common, "Given")).start();
      new Thread(new NewThread2(common, "Taken")).start();
    }
 }
```

Τι παρατηρείτε κατά την εκτέλεση του προγράμματος και ποιες οι διαφορές με το πρόγραμμα Java041 που είδαμε σε αυτό το κεφάλαιο;

ΚΕΦΑΛΑΙΟ 6ο

6. ΕΞΑΙΡΕΣΕΙΣ

6.1 ΕΙΣΑΓΩΓΗ

Οπωσδήποτε, τα λάθη στη διάρκεια εκτέλεσης προγραμμάτων (**run-time errors**) είναι από τους πρωταρχικούς στόχους για τον ορθό σχεδιασμό και ανάπτυξη λογισμικού εφαρμογών. Εδώ, η βασική ιδέα είναι να υπάρχει η ικανότητα λειτουργίας μεθόδων, ακόμη και σε περιπτώσεις που ο κώδικας αδυνατεί να ανταποκριθεί σωστά σε ορισμένες συνδυαστικές δεδομένων ή και γεγονότων που συμβαίνουν στη διάρκεια της εκτέλεσης. Το αποτέλεσμα μιας τέτοιας δυσλειτουργίας είναι κατά κανόνα η εμφάνιση **διαστολικού σημείου αυτοκατάρρευσης** του προγράμματος.

Οι αντικειμενοστρεφείς γλώσσες προγραμματισμού (Java και C#) όπως και η C++/Visual C++, εφαρμόζει κατά κανόνα τον (3) τρόπο των εξαιρέσεων. Οι εξαιρέσεις (**exceptions**) είναι γεγονότα (**events**), που συμβαίνουν στη διάρκεια εκτέλεσης προγράμματος και τα οποία παρεμποδίζουν την ομαλή εκτέλεση του προγράμματος. Κάθε φορά που συμβαίνει ένα τέτοιο γεγονός, η μέθοδος που εκτελείται, δημιουργεί ένα αντικείμενο εξαίρεσης, το οποίο και στέλνει στο διερμηνέα της γλώσσας. Ο τελευταίος, μόλις λάβει ένα τέτοιο αντικείμενο, προσπαθεί να βρει κατάλληλο κώδικα ώστε να αντιμετωπίσει το λάθος. Η όλη αυτή διαδικασία δημιουργίας αντικειμένων εξαίρεσης, παράδοσης αυτού στον διερμηνέα και συλλογή κώδικα, καλείται **ρίψη εξαίρεσης** (**exception throwing**). Ο διερμηνέας, για να βρει κατάλληλο κώδικα διαχείρισης της εξαίρεσης (**exception handler**), εξετάζει με αντίστροφη χρονολογική σειρά την στοίβα κλήσεις της μεθόδου που έστειλε την εξαίρεση. Αν βρεθεί τέτοιος κώδικα, τότε λέμε πως ο κώδικας διαχείρισης **συνέλαβε την εξαίρεση (catches the exception).** Αν όμως η στοίβα κλήσεων δεν περιέχει κώδικα διαχείρισης της εξαίρεσης, τότε η εκτέλεση του προγράμματος τερματίζεται.

6.2 ΠΑΡΕΧΟΜΕΝΕΣ ΚΛΑΣΕΙΣ ΕΞΑΙΡΕΣΕΩΝ

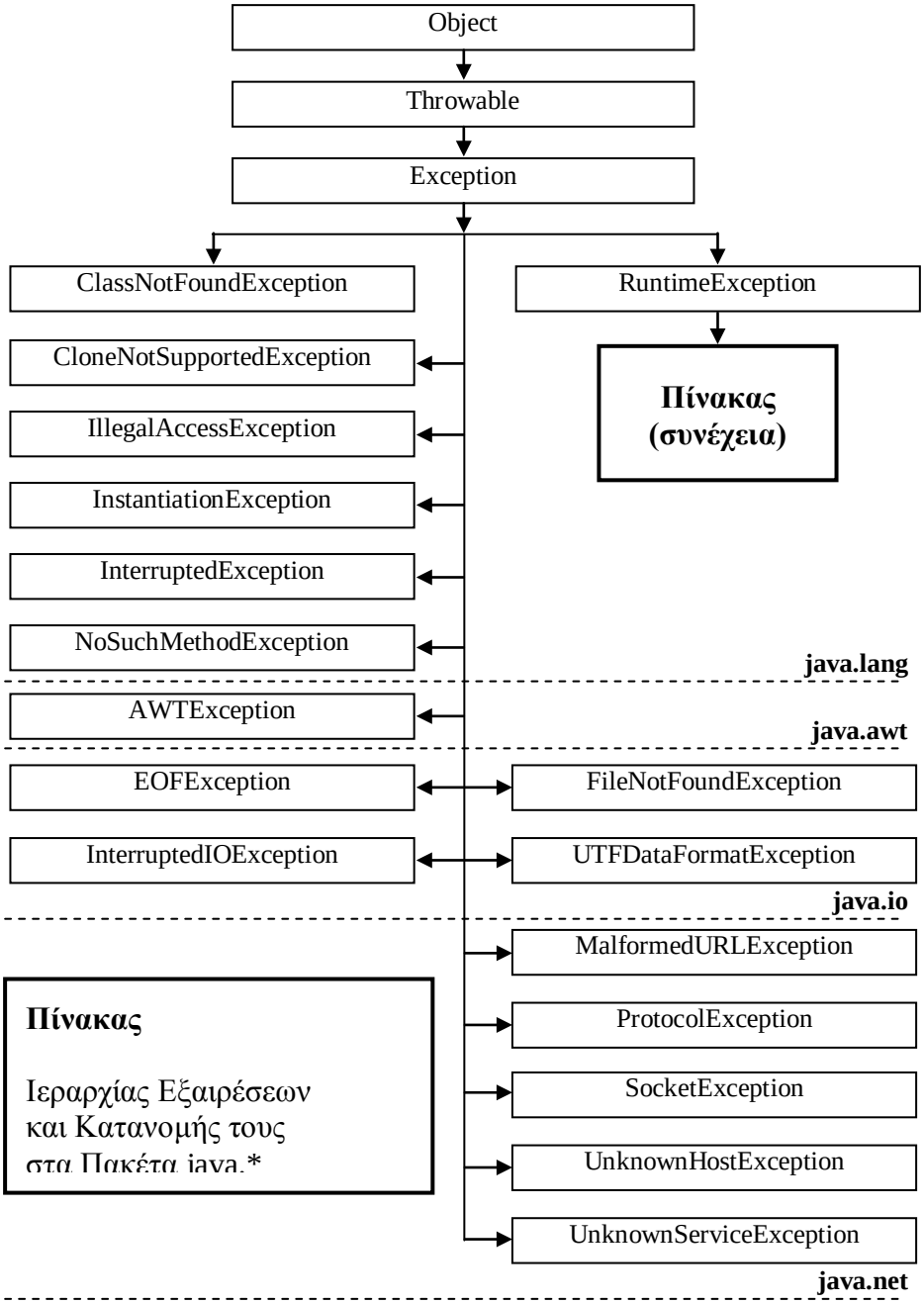
Οι αντικειμενοστρεφείς γλώσσες προγραμματισμού περιέχουν ενσωματωμένη διαχείριση των εξαιρέσεων μέσα από παρεχόμενες κλάσεις που τις συνοδεύουν. Διακρίνουν δύο κύριες κατηγορίες που η εκτέλεση του προγράμματος δεν μπορεί (ή δεν πρέπει) να συνεχισθεί φυσιολογικά:

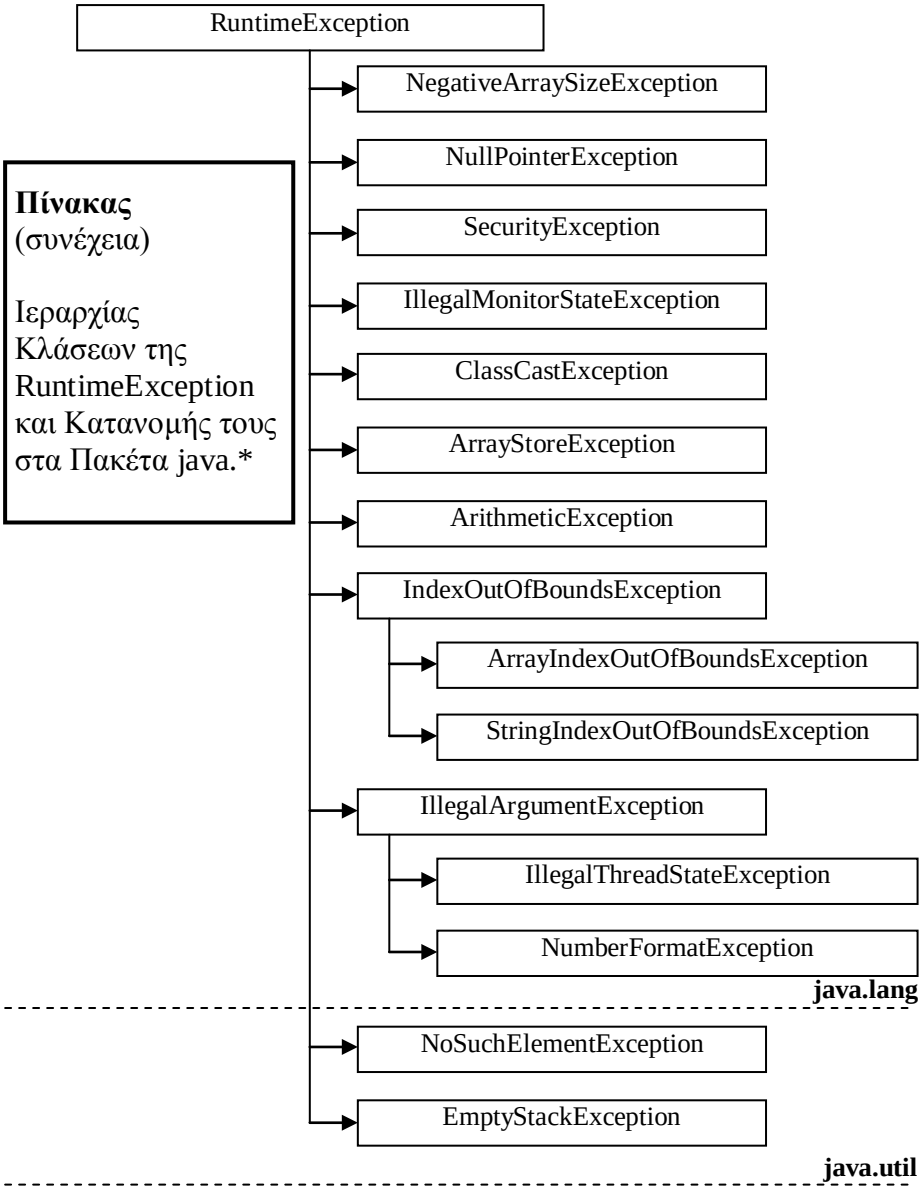
- **Εξαιρέσεις**: Ειδικές περιπτώσεις σφαλμάτων εκτέλεσης που μπορούν να ξεπερασθούν με κατάλληλη πρόβλεψη και χειρισμό.
- **Σφάλματα** (errors): Αφορά σφάλματα εκτέλεσης από τα οποία ο διερμηνέας της JAVA/C# είναι αδύνατον να συνεχίσει.

Οι εξαιρέσεις είναι αντικείμενα που προέρχονται από την κλάση **Exception** ή και από υποκλάσεις αυτής. Στον Πίνακα ιεραρχίας εξαιρέσεων στο API της JAVA και κατανομής τους στα πακέτα αυτής, φαίνονται τα ονόματα των παρεχομένων κλάσεων, αλλά και πού ανήκουν αυτές για να μπορεί ο Χρήστης να ρυθμίσει τις εντολές import του προγράμματός του. Ειδικά για τις υποκλάσεις της **Runtime Exception** ακολουθεί δεύτερος ειδικός Πίνακας με τα αντίστοιχα πακέτα αυτών.

Ο Πίνακας ιεραρχίας κλάσεων σφαλμάτων αφορά λάθη στη διάρκεια της εκτέλεσης, από τα οποία ο Διερμηνέας της JAVA είναι αδύνατον να ανανήψει. Όταν ένα τέτοιο σφάλμα εμφανισθεί, το πρόγραμμα διακόπτεται με **διαστολικό σημείο αυτοκατάρρευσης** με παράλληλη εμφάνιση διαγνωστικού μηνύματος.

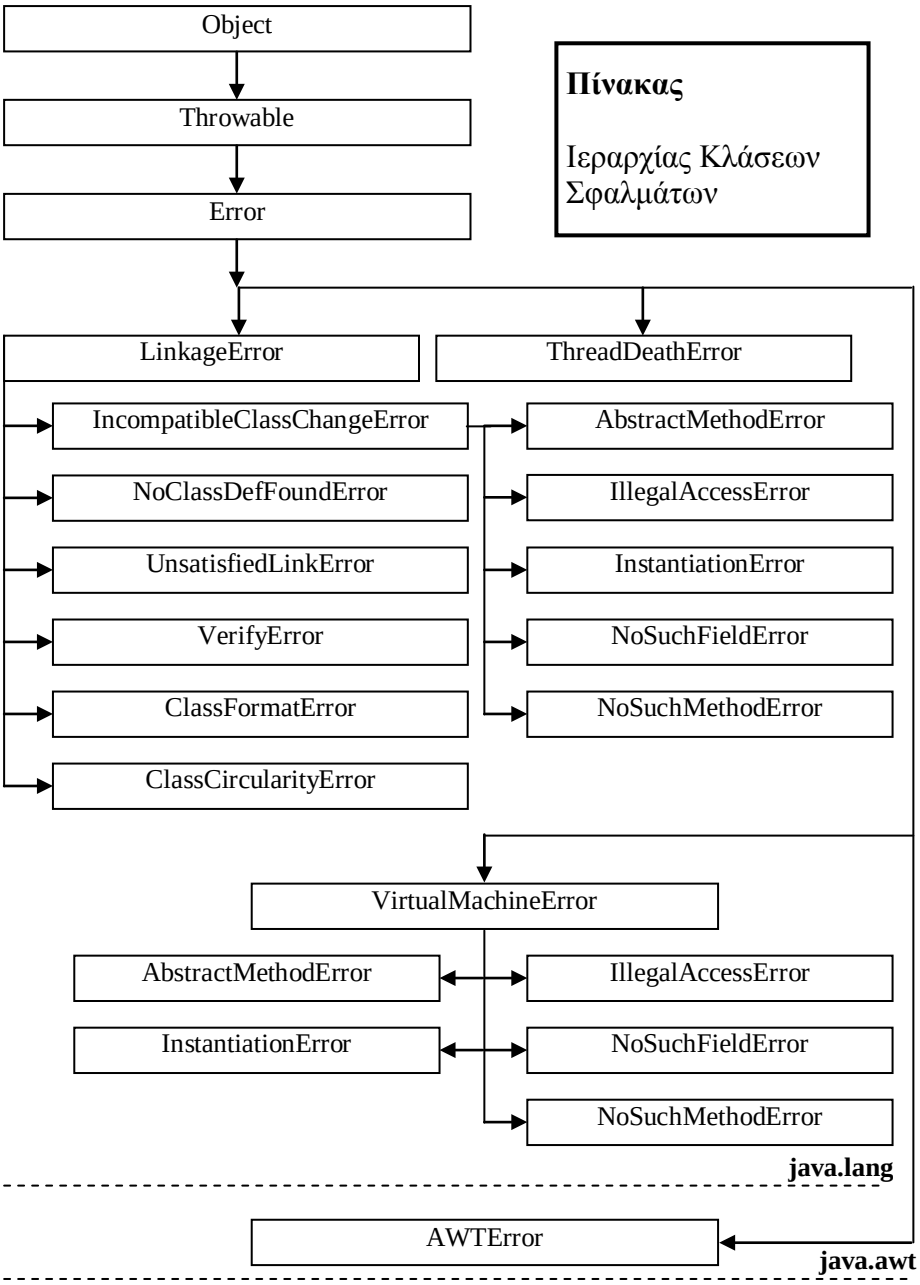
Αμέσως μετά από τους αντίστοιχους πίνακες στη Java, παρουσιάζεται η ιεραρχία των εξαιρέσεων στη C#. Ειδικότερα, παρουσιάζονται ορισμένες από τις ιεραρχίες της **System** καθώς και ορισμένες εξαιρέσεις της **System.IO**


Object
Throwable
Exception
ClassNotFoundException
RuntimeException
CloneNotSupportedException
IllegalAccessException
InstantiationException
InterruptedException
NoSuchMethodException
Πίνακας (συνέχεια)
java.lang
AWTException
java.awt
EOFException
FileNotFoundException
InterruptedIOException
UTFDataFormatException
java.io
MalformedURLException
ProtocolException
SocketException
UnknownHostException
UnknownServiceException
java.net
Πίνακας
Ιεραρχίας Εξαιρέσεων
και Κατανομής τους
στα Πακέτα java.*



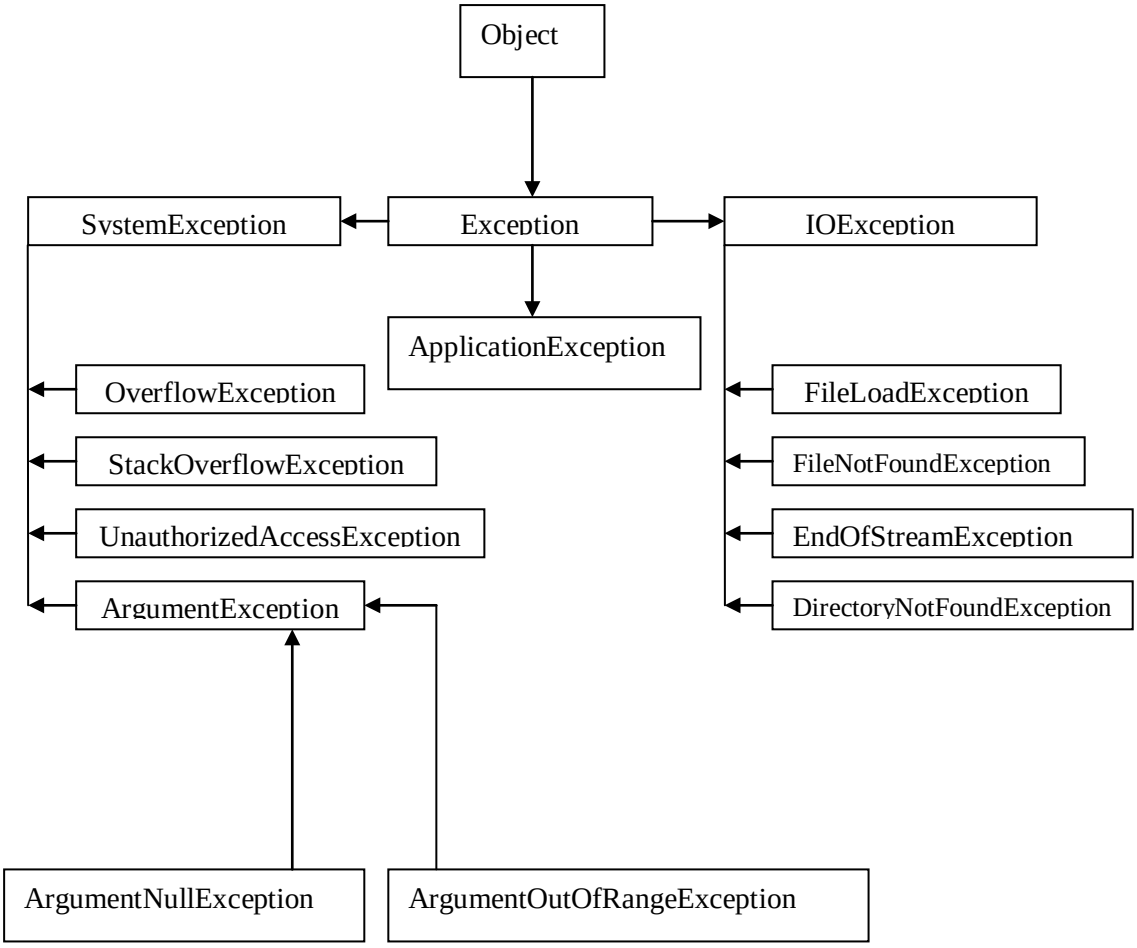
Πίνακας
(συνέχεια)

Ιεραρχίας Κλάσεων της RuntimeException και Κατανομής τους στα Πακέτα java.*



Πίνακας

Ιεραρχίας Κλάσεων Σφαλμάτων



6.3 ΕΝΤΟΛΕΣ ΧΕΙΡΙΣΜΟΥ ΕΞΑΙΡΕΣΕΩΝ

Ο **χειρισμός εξαιρέσεων (exception handling)** είναι κατά το πλείστον όμοιος στην Java και C#, ωστόσο οι δηλώσεις εξαίρεσης δεν είναι αναγκαίες για τη C#. Η C# πρόσθεσε εντολές χρησιμεύοντας στο να ελέγχουν πως η υπερχείλιση ακεραίων αριθμητικών θα χειρίζεται κατά το χρόνο μεταγλώττισης και κατά το χρόνο εκτέλεσης. Στον πίνακα παρακάτω, περιγράφονται οι σχέσεις των εντολών χειρισμού εξαιρέσεων.

Πίνακας Σύγκρισης εντολών χειρισμού εξαιρέσεων

Java	C#	Σχόλια
try	Try	Περικλείει ένα βρόγχο από εντολές. Εάν μια εξαίρεση προκληθεί από μια από αυτές τις εντολές η εκτέλεση του προγράμματος θα μεταφερθεί στην εντολή περιβαλλόμενη από ένα κατάλληλο βρόγχο catch.
catch	Catch	Η C# υποστηρίζει γενική και ανώνυμη πρόταση catch.
finally	Finally	Κώδικας για να εκτελέσει, αδιάφορο εάν υπάρχει ένας try…catch βρόγχος ή μέσω μια εξαίρεσης.
throw	Throw	Προκαλεί μια εξαίρεση.
throws	N/A	Δεν υποστηρίζεται στην C# .Για να δηλώσετε ότι κάποιος κώδικας στο σώμα της μεθόδου σα μπορεί να προκαλέσει μια εξαίρεση, προσθέστε απλώς την

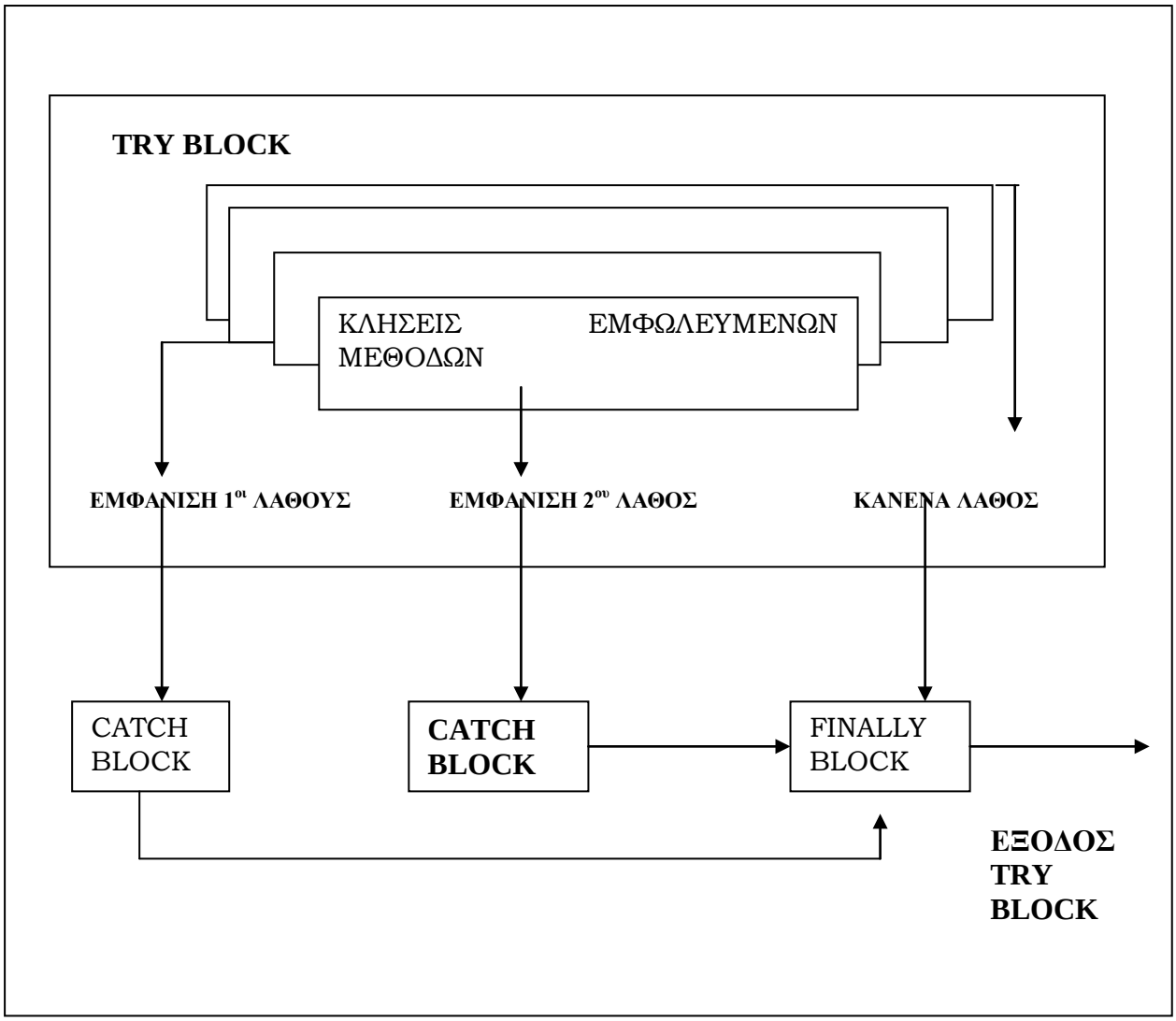
		εντολή throws.
N/A	Checked	Ενεργοποιεί τον έλεγχο της υπερχείλισης ακέραιων αριθμητικών.
N/A	Unchecked	Απενεργοποιεί τον έλεγχο της υπερχείλισης ακέραιων αριθμητικών

Ο προγραμματιστής όταν προβλέπει ότι ένα τμήμα του κώδικα είναι ύποπτο λαθών, δηλαδή μπορεί να προκαλέσει μία ή και περισσότερες εξαιρέσεις, εφαρμόζει τη δομή των εντολών **try** και **catch** με συντακτικό τύπο τον ακόλουθο, όπου η εντολή **finally** είναι προαιρετική:

```
try
        { Επικράτεια ύποπτου κώδικα;}
catch (Εξαίρεση1 αντικείμενο1)
        { Εντολές αντίδρασης της Εξαίρεσης1;}
catch (Εξαίρεση2 αντικείμενο2)
        { Εντολές αντίδρασης της Εξαίρεσης 2;}
.....
finally
        { Εντολές που θα εκτελεστούν μετά το try;}
// θα εκτελεσθούν σε κάθε περίπτωση
```

όπου όταν η ροή του προγράμματος μπει μέσα στην Επικράτεια του ύποπτου κώδικα του try, γίνεται έλεγχος για σφάλμα. Αν δεν εμφανισθεί κάποιο τέτοιο λάθος, το πρόγραμμα συνεχίζει κανονικά την εκτέλεση των Εντολών της εντολής finally και αν δεν υπάρχει, η διαδικασία συνεχίζεται προς τις εντολές που είναι εκτός και του τελευταίου catch. Διαφορετικά, προκαλείται μια εξαίρεση και η εκτέλεση διακόπτεται όσον αφορά τις υπόλοιπες εντολές του try. Ο έλεγχος μεταφέρεται σε κάποιο catch που χειρίζεται τύπο εξαίρεσης συμβατό με αυτήν που εμφανίσθηκε, οπότε εκτελούνται μόνον οι εντολές αυτού του catch. Σε κάθε περίπτωση, θα εκτελεσθούν στη συνέχεια και οι εντολές του finally (αν υπάρχει). Αν η εξαίρεση που προκαλείται σε ένα τμήμα try δεν είναι συμβατή με κανέναν τύπο που ορίζονται στα τμήματα catch, τότε η εξαίρεση μεταδίδεται προς τον υπόλοιπο κώδικα προς τα έξω μέχρις ότου συλληφθεί από κάποια εξωτερική δομή try/catch. Τελικά, αν η εξαίρεση μείνει ασύλληπτη, καταλήγει στον διερμηνέα της C# να τερματίζει την εκτέλεση με αντίστοιχα μηνύματα για όλα τα σημεία που η εξαίρεση έμεινε ελεύθερη!

Τα παραπάνω, γίνονται εύκολα κατανοητά από στο παρακάτω σχήμα:



Στην συνεχεία παρουσιάζεται ένα απλό πρόγραμμα Java όπου ο χρήστης δίνει 5 ορίσματα και αυτά εμφανίζονται στην οθόνη. Το όνομα του προγράμματος είναι Java015. Γίνεται έλεγχος των ορισμάτων που δίνει ο χρήσης. Αν ο αριθμός είναι μεγαλύτερος του 5 τότε εμφανίζεται το **Exception ArrayIndexOutOfBoundsException** που σημαίνει ότι έχουμε περάσει τα όρια του πίνακα προσθέτοντας παραπάνω τιμές-ορίσματα. Το πρόγραμμα χειρίζεται αυτό το Exception με την εντολή catch και στην συνεχεία οδηγούμαστε στην εντολή finally.

```
class Java015
 {  public static void main(String x[])
    { int i=0;
      try
       {  for(i=0; i<5 ;i++)
             System.out.println(x[i]);
       }
      catch(ArrayIndexOutOfBoundsException
      exception_object)
      {  System.out.println("Error : less than 5 arguments");
         System.out.println("Name of Exception: "
          +exception_object);
```

```
        }
        finally
        {   System.out.println("Last value of i="+i);
        }
        System.out.println("normal End...");
      }
    }
```

Ας δούμε τώρα το αντίστοιχο πρόγραμμα σε C#.

```
using System;

class Java015
 {  public static void  Main(System.String[] x)
    {  int i = 0;
       try
        { for (i=0; i < 5; i++)
            System.Console.WriteLine(x[i]);
        }
       catch (System.IndexOutOfRangeException exception_object)
       { System.Console.WriteLine("Error : less than 5
         arguments");
         System.Console.WriteLine("Name of Exception: " +
         exception_object);
       }
       finally
       { System.Console.WriteLine("Last value of i=" + i);
       }
       System.Console.WriteLine("normal End...");
     }
 }
```

Παρατηρείστε ότι η δομή των δυο προγραμμάτων είναι η ίδια. Η λειτουργία των αντιστοίχων try,catch,finally επίσης η ίδια. Η μόνη αλλαγή είναι η αλλαγή ονομασίας του Exception που στην περίπτωση μας, στη C#, ονομάζεται **IndexOutOfRangeException**.

6.4 ΠΟΛΛΑΠΛΟΤΗΤΑ ΕΞΑΙΡΕΣΕΩΝ

Όπως ήδη αναφέρθηκε, σε μια λογισμική παράγραφο ενός try, είναι δυνατόν να υπάρχουν περισσότερα του ενός catch, τα οποία βέβαια διαχειρίζονται διαφορετικούς τύπους εξαιρέσεων (**multiple catch clauses**). Σε περίπτωση εμφάνισης εξαίρεσης, η διαχείριση μεταφέρεται σε εκείνο το catch που περιέχει συμβατό κώδικα με το είδος της εξαίρεσης που εμφανίσθηκε.

Στο πρόγραμμα Java016.java που ακολουθεί, ο δοκιμαζόμενος κώδικας του try, είναι φανερό πως μπορεί να προκαλέσει δύο εξαιρέσεις. Η κανονική εκτέλεση του προγράμματος πραγματοποιείται όταν στη γραμμή εκτέλεσης δοθούν πέντε τουλάχιστον παράμετροι, διαφορετικά υπάρχουν τα ακόλουθα catch:

(1) Για διαχείριση εξαίρεσης διαίρεσης με μηδέν, όταν δεν δοθεί παράμετρος στη γραμμή εκτέλεσης του προγράμματος (**Arithmetic Exception**), οπότε εμφανίζεται το μήνυμα που υπάρχει σε σχετική εντολή εξόδου:

```
System.out,println("Divide by zero...!");
```

(2) Αν δοθούν από μία μέχρι και τέσσερις παράμετροι στη γραμμή εκτέλεσης του προγράμματος, λειτουργεί η διαχείριση εξαίρεσης για δείκτες μητρών εκτός ορίων (**ArrayIndexOutOfBoundsException**), διότι μέσα στο try εμφανίζεται το στοιχείο x[4], το οποίο δεν έχει προβολή στην κύρια μνήμη για τοποθέτηση της τιμής του. Στην περίπτωση αυτή, έχουμε την εμφάνιση αντίστοιχου μηνύματος που περιέχει στο δεύτερο catch:

```
System.out,println("Error:less than 5 arguments");
```

```
// multiple Exceptions
class Java016
{   public static void main(String x[])
    {   int i=0,f=10;
        int args = x.length;
        try
        {   int y=f/args;
            for(; i<5 ;i++)
            Console.WriteLine(x[i]);
        }
        catch(ArithmeticException e1)
        {   Console.WriteLine("Divide by zero...!");
        }
        catch(ArrayIndexOutOfBoundsException e2)
        {   Console.WriteLine(
                "Error : less than 5 arguments");
        }
        finally
        {   Console.WriteLine("Last value of i="+i);
        }
        Console.WriteLine("normal END...");
    }
}
```

Στη C# το αντίστοιχο πρόγραμμα, είναι το ακόλουθο:

```
using System;
using System.Diagnostics;

namespace cs016
{ class Class1
```

```
    { static void Main(string[] args)
        {   int i=0;
            int f=10;
            int x=args.Length;
            try
            { int y=f/x;
              for (i=0;i<5;i++)
                { Console.WriteLine(args[i]);
                }
            }
            catch (System.ArithmeticException e)
            { Console.WriteLine("DIVIDE BY ZERO!!!!!!");
            }
            catch (System.IndexOutOfRangeException e1)
            { Console.WriteLine("ERROR: LESS THAN 5 ARGUMENTS");
            }
            finally
            { Console.WriteLine("LAST VALUE OF i = 0" +i);
            }
            Console.WriteLine("NORMAL END....");
          }
     }
}
```

Ακολουθεί ένα παράδειγμα εξόδου του προγράμματος Java016 στην κανονική περίπτωση και στις δύο περιπτώσεις εμφάνισης εξαιρέσεων ίδια ακριβώς είναι και η έξοδος του προγράμματος στη C#:

```
java Java016 1 2 3 4 5
1
2
3
4
5
Last value of i=5
normal End...

java Java016
Divide by zero...!
Last value of i=0
normal End...

java Java016 1 2
1
2
Error : less than 5 arguments
Last value of i=2
normal End...
```

Είναι δυνατόν να υπάρχουν φαινόμενα αλυσιδωτά με τη δομή των εντολών try και catch (**nesting try phenomenon**). Σε τέτοιες περιπτώσεις σε όποιο επίπεδο εμφανισθεί εξαίρεση, γίνεται προσπάθεια διαχείρισης αυτής από κάποιο catch του επιπέδου αυτού. Αν δεν βρεθεί κάποιο συμβατό τέτοιο catch, τότε γίνεται έρευνα για συμβατό catch στο αμέσως εξωτερικό try. Βέβαια, ένας κώδικας με τέτοια αλυσιδωτά φαινόμενα, απέχει πολύ από αυτό που ονομάζουμε αριστοποιημένο πρόγραμμα.

6.5 ΠΡΟΚΛΗΣΗ ΕΞΑΙΡΕΣΕΩΝ

Μέχρι τώρα, η σύλληψη των εξαιρέσεων γινόταν από το run time system της C#. Ωστόσο, μια εξαίρεση μπορεί να προκληθεί από το χρήστη με την ειδική εντολή **throw** με συντακτικό τύπο:

```
throw αντικείμενο_Exception
```

ή

```
throw υποκλάση _Exception
```

όπου η παράμετρος, μπορεί να είναι αντικείμενο της κλάσης Exception ή κάποια υπό-κλάση αυτής. Στην πράξη, το throw χρησιμοποιείται με την βοήθεια του τελεστή **new** για τον προσδιορισμό κάποιου στιγμιότυπου κάποιας εξαίρεσης ή/και μέσα στα πλαίσια κάποιας εντολής try/catch περνώντας συνήθως κάποια παράμετρο στη διαχείριση της εξαίρεσης. Ωστόσο, όταν το throw υπάρχει χωρίς τις εντολές try/catch το πρόγραμμα μετά το throw, διακόπτεται με αντίστοιχο μήνυμα. Ακολουθεί ένα παράδειγμά για την καλύτερη κατανόηση της εντολής, το Proklisi.java.

```
class Proklisi
{ public static void main(String x[])
{ int i=0;
int len = x.length;
if (len<5)
{
 throw new ArrayIndexOutOfBoundsException("incomplete number of
   arguments ….");
}
for (i=0;i<5;i++)
{
System.out.println("x[i]")
}
System.out.println("Normal End");}}
```

για την C# το ίδιο πρόγραμμα συντάσσεται ως εξής:

```
using System;

namespace Proklisi
{ class Class1
    { static void Main(string[] args)
      { int i=0;
```

```
            int len = args.Length;
            if (len<5)
             {
              throw new IndexOutOfRangeException("incomplete number of
              arguments ...");
             }
            for (i=0;i<5;i++)
             {
              Console.WriteLine(args[i]);
             }
            Console.WriteLine("Normal End");
            }
        }
    }
```

παρατηρούμε ότι και στα δυο προγράμματα, η εκτέλεσή τους διακόπτεται όταν δοθούν λιγότερες από πέντε παραμέτρου, αφού με την throw πιάνουμε το exception κατά τη διάρκεια εκτέλεσης του προγράμματος.

6.6 ΥΠΟΚΛΑΣΕΙΣ ΕΞΑΙΡΕΣΕΩΝ ΧΡΗΣΤΩΝ

Τόσο η Java, όσο και η C#, προσφέρουν μια ικανοποιητική συλλογή εξαιρέσεων και σύλληψης αυτών όπως έχει ήδη αναφερθεί. Ωστόσο, υπάρχει η δυνατότητα ορισμού νέων υποκλάσεων εξαιρέσεων από το χρήστη. Σε μια τέτοια περίπτωση, κάθε νέα υποκλάση, θεωρείται ότι ανήκει στην υποκλάση **exception.**

Στον παρακάτω πίνακα συγκρίνονται οι λειτουργικότητες των κλάσεων System.Exception από το .NET με τις κλάσεις java.lang.Throwable από τη Java.

Πίνακας Σύγκρισης των Java και C# μελών εξαίρεσης

Java	C#	Σχόλια
Throwable()	Exception()	Κατασκευάζει μια νέα περίπτωση εξαίρεσης.
Throwable(String)	Exception(String)	Κατασκευάζει μια νέα εξαίρεση με ορισμένο μήνυμα.
Throwable(String,Throwable)	Exception(String,Exception)	Κατασκευάζει μια νέα εξαίρεση με ορισμένο μήνυμα κι αλυσιδωτές εξαιρέσεις.
N/A	Exception(SerializationInfo, StreamingContext)	Κατασκευάζει μια νέα εξαίρεση από μια πρώην δημοσιευόμενη εξαίρεση. Δείτε GetObjectData παρακάτω στο πίνακα.
fillStackTrace()	N/A	Δεν υποστηρίζεται. Όμως μπορεί το Enviroment.StackTrac

		e να χρησιμοποιηθεί μόνο το διαγνωστικό στοίβας χρειάζεται.
getCause()	InnerException	Παίρνει τη περίπτωση εξαίρεσης ή null εάν ένα έχει οριστεί.
getLocalizedMessage()	N/A	Το .NET προτείνει το μήνυμα να εντοπίζεται διαμέσου της ιδιότητας του Message. Η Java παρέχει άλλη μέθοδο η οποία υπερκαλύπτεται από την παραγόμενη κλάση, να επιστρέφει την εντοπισμένη έκδοση του μηνύματος.
getMessage()	Message	Δίνει ένα μήνυμα που περιγράφει την εξαίρεση.
getStackTrace()	StackTrace	Δίνει το διαγνωστικό στοίβας σαν string από την εξαίρεση.
initCause()	N/A	Ρυθμίζει τις αναφορές για τις αλυσιδωτές εξαιρέσεις. Το .NET επιτρέπει αυτή τη αναφορά να ρυθμίζεται κατά τη κατασκευή.
printStackTrace()	N/A	Δεν υποστηρίζεται. Χρησιμοποιείστε τη μέθοδο ToString ακολουθούμενη από τις μεθόδους Stream ή Console.
setStackTrace()	N/A	Δεν υποστηρίζεται από τη C#.
N/A	HelpLink	Δίνει ή ορίζει μια σύζευξη για να δίνει πληροφορίες βοήθειας. Πρέπει να είναι μέσα στη φόρμα μια URN ή URL.
N/A	Source	Δίνει ή ορίζει το όνομα της εφαρμογής ή του αντικειμένου που προκαλεί εξαίρεση.
N/A	TargetSite	Δίνει μια MethodBase κλάση παρουσιάζοντας τη μέθοδο η οποία ήταν που προκάλεσε την εξαίρεση.
N/A	GetBaseException()	Επιστρέφει την αρχική

		εξαίρεση στη βασική αλυσιδωτή εξαίρεση.
N/A	GetObjectData()	Δημοσιεύει μια εξαίρεση.
N/A	HResult	Δίνει ή ορίζει ένα κωδικό αριθμό που αναγνωρίζει μοναδικά μια εξαίρεση. Κατά το πλείστον για ένωση κληρονομιάς.

6.7 ΟΙ ΕΝΤΟΛΕΣ CHECKED ΚΑΙ UNCHECKED

Αριθμητική υπερχείλιση λαμβάνει χώρα όταν τα αποτελέσματα ενός τύπου integer σε μια μαθηματική πράξη ή μετατροπή είναι μεγαλύτερα ή μικρότερα από το πεδίο τιμών που μπορεί να καλύψει. Για παράδειγμα, στη Java και στη C# ο τύπος τιμών short μπορεί να καταχωρήσει ένα δεκαδικό μεταξύ –32768 και 32767. Το επόμενο παράδειγμα αποδεικνύει αριθμητική υπερχείλιση στη Java :

```
short x = 32767;
x++;
System.out.println(x);
```

Το εξαγόμενο αποτέλεσμα από τον παραπάνω κώδικα θα είναι –32768.

Η Java χειρίζεται τις αριθμητικές υπερχειλίσεις των δεκαδικών περικόπτοντας το δυαδικό ψηφίο ανώτερης τάξης (high-order bit), με αποτέλεσμα τη κυκλική ενέργεια όπως αποδείχθηκε στο παραπάνω παράδειγμα.

Τα keywords **checked και unchecked** χρησιμοποιούν την ίδια σύνταξη. Ένα παράδειγμα checked σαν σύμβολο και σαν εντολή είναι το παρακάτω :

```
// checked as operator
int num = checked (x  * 3)

// checked as statement block
checked {
      int a = x++;
      int b = x * 3;
      int c = a * b ;
}
```

Ένα λάθος από το μεταγλωττιστή θα εμφανιστεί εάν η τιμή από το σύμβολο checked μπορεί να αποφασιστεί κατά το χρόνο μεταγλώττισης και η υπερχείλισης έχει συμβεί. Εάν η τιμή από τη checked διατύπωση αδυνατεί να καθοριστεί ως εκείνη τη στιγμή, η υπερχείλιση προκαλεί το **System.OverflowException** να απορριφθεί. Η χρήση του unchecked keyword επιβάλλει περικοπές, προκαλώντας κάθε λάθος ή δηλώσεις να

καταστέλλονται και το αποτέλεσμα να είναι η ίδια κυκλική συμπεριφορά όπως στη Java.

Εάν ούτε το checked μήτε το unchecked δεν είναι ορισμένα στο κώδικα τότε οι επόμενες προκαθορισμένες ενέργειες θα γίνουν :

- Για σταθερές τιμές, ο μεταγλωττιστής ελέγχει για υπερχείλιση και ένα λάθος θα εμφανιστεί, όπως θα γινόταν με το checked keywords.

- Για μεταβλητές τιμές, ο χρόνος εκτέλεσης (runtime) θα επιτρέψει περικομμένες τιμές, όπως θα γινόταν με το unchecked keywords.

6.8 ΑΣΚΗΣΕΙΣ

1. Να γραφεί πρόγραμμα υπολογισμού του ελαχίστου στοιχείου μήτρας nxm, με max(n,m)<=500, ακεραίων τιμών.

```
// Program Java010.java
//Reading parameters from keyboard (standard input)
public class Java010
   { public static double input(String s)
      { byte buffer[] = new byte[100];
      System.out.print(s + ": ");
      try
         {
         System.in.read(buffer);
         }
      catch (java.io.IOException e)
         {
         System.out.println("Exception " +
      "encountered: " + e.getMessage());
         System.exit(1);
         }
      return(new Double((new
      String(buffer)))).doubleValue();
      }
   // main method
   public static void main(String args[]){
      int lines,verticals,min=0;
      double work;
      int array[][]=new int[500][500];
      do{
         work =input(
         "Give the number of rows (<=500) ==>");
         lines = (int) work;
         work =input(
         "Give the number of columns (<=500)==>");
         verticals = (int) work;}
      while ((lines>verticals?lines:verticals)>500
```

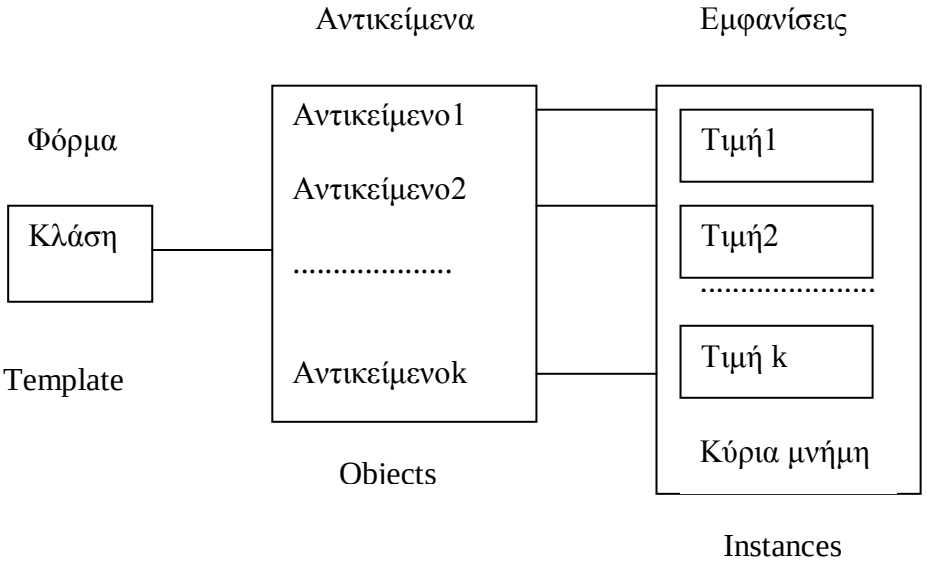
```
    ||(lines>verticals?lines:verticals)<=0);
for(int i=0; i<lines; i++)
    for(int j=0; j<verticals; j++)
        {System.out.print(
        "for the element ["+i+"]["+j+"] ");
        work = input("==>");
        array[i][j]=(int) work;
        if((i==0)&&(j==0)) min=array[0][0];
        min=array[i][j]<min?array[i][j]:min;
        }
System.out.println("Minimum Element ="+min);
}}
```

ΚΕΦΑΛΑΙΟ 7ο

7. ΚΛΑΣΕΙΣ ΚΑΙ ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ

7.1 ΕΙΣΑΓΩΓΗ

Το μοντέλο της αντικειμενοστραφούς τεχνολογίας ανάπτυξης λογισμικού (**Object Oriented Software Development**) έχει σα βασικό στόχο το σχεδιασμό και την ανάπτυξη **προγραμματιστικών ολονίων** που να μπορούν να μεταγλωττίζονται ανεξάρτητα από την εφαρμογή, **να ομαδοποιούν μεταβλητές και μεθόδους** (συναρτήσεις ή και υποπρογράμματα) και να είναι δυνατόν να οριστούν **σύνθετες μεταβλητές** πάνω σε όλο το προγραμματιστικό ολόνιο. Ένα τέτοιο προγραμματιστικό ολόνιο, ονομάζεται **κλάση (class)** και οι σύνθετες μεταβλητές που ορίζονται πάνω της, ονομάζονται **αντικείμενα (objects).** Οι μεταβλητές που εμφανίζονται σαν μέλη μιας κλάσης, ονομάζονται **στιγμιαίες μεταβλητές (instance variables),** ενώ οι συναρτήσεις και υποπρογράμματα μιας κλάσης, ονομάζονται **μέθοδοι (methods)** επομένως, η κλάση είναι η φόρμα για το αντικείμενο και το δεύτερο είναι μια **εμφάνιση (στιγμιότυπο)** της κλάσης **(template/instance).** Τελικά, όμως, εκείνο που θα έχει προβολή στην κύρια μνήμη και δέσμευση αυτής, για φιλοξενία τιμών θα είναι τα αντικείμενα ενώ η κλάση θα υπάρχει σαν μέρος του κώδικα που εκτελείται.



Επομένως, οι κλάσεις είναι κατηγορίες αντικειμένων με την ίδια συμπεριφορά. Τα περιεχόμενα μιας κλάσης (class members) ονομάζονται **μέλη της κλάσης** και μπορούν να είναι μεταβλητές στιγμιαίες, που χαρακτηρίζουν την κατάσταση των αντικειμένων ή και μέθοδοι που χαρακτηρίζουν τη συμπεριφορά τους.

7.2 ΔΗΛΩΣΕΙΣ ΣΤΑΘΕΡΩΝ

Ο **Readonly Modifier** της C# είναι ισοδύναμο με τα constant της Java final static. Το βασικό κύτταρο, είναι η κλάση στον αντικειμενοστρεφή προγραμματισμό. Εκτός αυτό είναι το enum, struct. Σημαντική διαφορά μεταξύ της Java και της C# υπάρχει σε επίπεδο σταθερών μεταβλητών των κλάσεων. Στη Java η λέξη –κλειδί **final** αρκεί για να δηλωθεί ότι μια μεταβλητή είναι τελικώς σταθερή μέσα σε μια κλάση, δηλαδή η τιμή της δίνεται a priori στη διάρκεια του compilation και δεν διαμορφώνεται στη διάρκεια της εκτέλεσης. Στη C# οι σταθερές δεν περιορίζονται απλά με μια δήλωση όπως τη Java, αλλά πρόκειται στην ουσία περί ειδικού τύπου δεδομένου που δηλώνεται με τη λέξη κλειδί **const**. Οι δηλώσεις σταθερών στη C#, έχουν γενικό συντακτικό τύπο:

[attributes] [modifiers] const type identifier =value;

Μερικά παραδείγματα είναι τα παρακάτω.

```
public const int i=60;
protected const string name =”Pavlos” surname= null;
protected internal const int j=j*2;
```

οι μόνες τιμές που επιτρέπεται να πάρουν οι σταθερές της C# είναι **string, literals, null**.

Ο προσδιοριστής (modifier) μιας σταθερής μεταβλητής για μια κλάση ή ένα struct παίρνει τις ακόλουθες τιμές:

	Member of class	Member of struct
Public	√	√
Protected	√	N/A
Private	√	√
Internal	√	√
protected internal	√	N/A
New	√	N/A
Abstract	N/A	N/A
Sealed	N/A	implicit
Virtual	N/A	N/A
Override	N/A	N/A
Readonly	N/A	N/A
Volatile	N/A	N/A
Static	Implicit	N/A
Extern	N/A	N/A

Όπου:

1. Με το όρο **public** δηλώνουμε πως η σταθερά της κλάσης είναι προσπελάσιμη από οποιοδήποτε σημείο μέσα στη κλάση ή και εκτός αυτής.
2. Με τον όρο **protected** η σταθερά της κλάσης είναι προσπελάσιμη από κλάσεις που είναι υποκλάσεις της κλάσης που ορίζεται.
3. Η δήλωση **private** σημαίνει πως η σταθερά της κλάσης είναι προσπελάσιμη μόνο μέσα στην κλάση που ορίζεται

4. Με τον όρο **internal** εννοούμε πως η σταθερά της κλάσης είναι προσπελάσιμη από οποιαδήποτε κλάση εντός της επικράτειας της κλάσης που ορίζεται
5. Με τον όρο **protected internal** η σταθερά της κλάσης είναι προσπελάσιμη από οποιαδήποτε κλάση εντός της επικράτειας της κλάσης που ορίζεται ή από κλάσεις που είναι υποκλάσεις της κλάσης που ορίζεται.
6. Με τον όρο **new** δεσμεύουμε μνήμη για την αποθήκευση της μεταβλητής.

7.3 ΜΕΤΑΒΛΗΤΕΣ ΚΛΑΣΗΣ

Όπως ήδη έχει αναφερθεί, η κλάση μπορεί να περιέχει στιγμιαίες μεταβλητές και μεθόδους. Ο γενικός συντακτικός τύπος δήλωσης μιας μεταβλητής κλάσης είναι ο ακόλουθος:

[modifiers] type identifier = value;

μερικά παραδείγματα είναι τα παρακάτω.

```
public int i;
public byte x=5,z=100;
```

Οι τιμές του προσδιοριστή modifier, φαίνονται στον παρακάτω πίνακα.

	Member of class	Member of struct
Public	√	√
Protected	√	N/A
Private	√	√
Internal	√	√
Protected internal	√	N/A
New	√	N/A
Abstract	N/A	N/A
Sealed	N/A	Implicit
Virtual	N/A	N/A
Override	N/A	N/A
Readonly	√	√
Volatile	√	√
Static	√	√
Extern	N/A	N/A

Όπου:

1. Όταν μια μεταβλητή δηλωθεί σαν **static** στην ουσία μετατρέπει το μέλος της κλάσης **σε σφαιρική μεταβλητή (global variable).** Το όνομα των μεταβλητών είναι γνωστό είτε μόνο του εντός της κλάσης αυτού, είτε με τη βοήθεια το ονόματος της κλάσης του:

   Ονομα_κλάσης. Ονομα_μεταβλητής

   Η δήλωση static, οδηγεί τον κώδικα σε προσδιορισμό πεδίων κύριας μνήμης της μεταβλητής αυτής πριν καν οριστούν τα αντικείμενα του

προγράμματος.

2. Όταν μια μεταβλητή δηλωθεί σαν **Volatile,** ταυτίζεται η δήλωση αυτή με εκείνη της Java και καθορίζει αν ένα μέλος κλάσης μπορεί να μεταβάλει την τιμή του ασύγχρονα από δύο ή και περισσότερες νηματικές εργασίες της ίδιας μεθόδου (ή και από διαφορετικές) (multithreaded programs). Ο προσδιοριστής **ReadOnly** και ο προσδιοριστής **Volatile** είναι **αμοιβαίως αποκλειόμενοι**

Όπως είναι φανερό, ένα μέλος κλάσης μπορεί να προσπελαστεί στο εσωτερικό της κλάσης, απλά με το όνομά του. Όταν μια μεταβλητή αναφέρεται μέσα σε μια μέθοδο που ανήκει στην κλάση που ορίζεται, είναι ορατή και γνωστή απλά με το όνομά της. Ωστόσο, και οι τοπικές μεταβλητές της μεθόδου είναι γνωστές μέσα στη μέθοδο με το όνομα τους. Τι θα συμβεί, όμως, αν μια τοπική μεταβλητή μεθόδου έχει το ίδιο όνομα με το μέλος της κλάσης; Σε κάθε κλάση υπάρχει ένας δείκτης που δείχνει στα ίδια τα αντικείμενα της κλάσης, με το όνομα **this** και επομένως μέσα στη μέθοδο μπορούμε να αναφερόμαστε στα συνώνυμα μέλη της κλάσης με τις τοπικές μεταβλητές με την απλή δήλωση:

This. όνομα_μέλους_κλάσης

Ένα απλό παράδειγμα για να μπορέσουμε να κατανοήσουμε τη λειτουργία του this είναι το παρακάτω. Στο συγκεκριμένο πρόγραμμα που ακολουθεί, μέσα στην κλάση MyClass υπάρχουν υποκατάστατες τοπικές μεταβλητές της μεθόδου rest, width και height οι οποίες έχουν τα ίδια ονόματα με δύο μέλη της ίδιας κλάσης, οπότε, εκεί που χρειάζεται να υπάρχου τα μέλη, χρησιμοποιείται το this. το πρόγραμμα δέχεται στη γραμμή εκτέλεσης, τέσσερις πραγματικές τιμές x1, x2, x3, x4 όπου τα x1, x2 είναι το μήκος και πλάτος ορθογωνίου και τα x3, x4 το μήκος και πλάτος εσωτερικού ορθογωνίου μέσα στο πρώτο. Η έξοδος του προγράμματος java020 συνίσταται στο υπόλοιπο εμβαδόν του εξωτερικού ορθογωνίου αφού αφαιρεθεί το εμβαδόν του εσωτερικού ορθογωνίου.

```
class   MyClass
 { double width;
   double height;
   double rest(double width, double height)
    {
      return (this.width)*(this.height)-(width*height);
    }
  }
class java020
 { public static void main(String x[])
   { double pw=0, ph=0;
     MyClass object= new MyClass();
     try
      { object.width=(new Double(x[0])).doubleValue();
        object.height=(new Double(x[1])).doubleValue();
        pw= (new Double(x[2])).doubleValue();
        ph= (new Double(x[3])).doubleValue();
      }
      catch(ArrayIndexOutOfBoundsException e1)
```

```
        { System.out.println("incomplete data...");
          return;
        }
     catch (NumberFormatException e2)
     { System.out.println("errors in data...");
       return;
     }
     System.out.println("rest area ="+object.rest(pw, ph));
     }
    }
```

Η ισοδύναμη μορφή του παραπάνω προγράμματος σε C# είναι η ακόλουθη:

```
using System;
class MyClass
{ internal double width;
  internal double height;
  internal virtual double rest(double width, double height)
   { return (this.width) * (this.height) - (width * height);
   }
}
class java020
 { [STAThread]
   public static void  Main(System.String[] x)
   { double pw = 0, ph = 0;
     MyClass object_Renamed = new MyClass();
     try{ object_Renamed.width = (System.Double.Parse(x[0]));
          object_Renamed.height = (System.Double.Parse(x[1]));
          pw = (System.Double.Parse(x[2]));
          ph = (System.Double.Parse(x[3]));
         }
    catch (System.IndexOutOfRangeException e1)
     { Console.WriteLine("incomplete data...");
       return ;
     }
     catch (System.FormatException e2)
      { Console.WriteLine("errors in data...");
        return ;
      }
     Console.WriteLine("rest area="+
      object_Renamed.rest(pw,ph));
   }
 }
```

Τρεις χαρακτηριστικές έξοδοι του προγράμματος είναι οι ακόλουθοι:

Java020 4 3 2 1
Rest area =10.0

Java020 s 3 2 1
Errors in data...

Java020 4 3
Incomplete data...

Η κλάση MyClass ορίζει αυτόματα τον τρόπο προσπέλασης των μελών της width και height. Ισοδύναμα, η κλάση αυτή θα μπορούσε να είχε υποστεί compilation χωριστά από την κλάση java020. Σε κάθε περίπτωση θα παραχθεί ο ενδιάμεσος κώδικας και των δύο κλάσεων. Βέβαια, η κλήση των μελών της κλάσης MyClass μέσα από την κλάση java020, γίνεται με τη βοήθεια του αντικειμένου object_Renamed.

7.4 ΜΕΘΟΔΟΙ ΚΛΑΣΗΣ

Όπως ήδη έχει αναλυθεί, κάθε συνάρτηση ή υποπρόγραμμα που περιγράφει τη συμπεριφορά αντικειμένων μέσα στον αντικειμενοστραφή προγραμματισμό, είναι ενσωματωμένο σε μια κλάση και ονομάζεται **μέθοδος (method)** ο γενικός συντακτικός τύπος της μεθόδου είναι:

[Modifiers] return type identifier (parameter list) { body }

Ένα απλό παράδειγμα ορισμού μεθόδου είναι το παρακάτω:

protected static double calc(int j, double b) { body code }

Οι τιμές του προσδιοριστή modifier, φαίνονται στον παρακάτω πίνακα.

	Member of class	Member of struct	Member of Interface
Public	√	√	Implicit
Protected	√	N/A	N/A
Private	√	√	N/A
Internal	√	√	N/A
protected internal	√	N/A	N/A
New	√	N/A	√
Abstract	√	N/A	Implicit
Sealed	√	N/A	√
Virtual	√	N/A	N/A
Override	√	N/A	N/A
Readonly	N/A	N/A	N/A
Volatile	N/A	N/A	N/A
Static	√	√	N/A
Extern	√	√	N/A

Αν δηλωθεί μια μέθοδος σαν **abstract** θα πρέπει να μην περιέχει το σώμα των εντολών της στην κλάση της. Σε τέτοια περίπτωση, θα πρέπει να υπάρχει το ανάπτυγμα (υλοποίηση της μεθόδου) σε κάποια υποκλάση, με το ίδιο όνομα της μεθόδου και τη λίστα παραμέτρων (ορισμάτων) που είχε αρχικά οριστεί η μέθοδος. Όταν μια μέθοδος είναι abstract τότε και η ίδια η κλάση θα

πρέπει να είναι abstract οπότε και δεν έχει νόημα ο ορισμός αντικειμένου πάνω στη κλάση αυτή.

Ο τύπος επιστρεφόμενης τιμής είναι ο τύπος τιμών που παίρνει η ίδια η μέθοδος. Αν η μέθοδος είναι **υποπρόγραμμα (subroutine)** τότε σαν τύπος τιμών τίθεται η λέξη κλειδί **void.** Προσοχή, οι ειδικές συναρτήσεις κλάσεων **(constructors)** δεν επιστρέφουν ούτε τιμή και δεν δέχονται σαν δήλωση το void. Η επιστρεφόμενη τιμή μεθόδου, υλοποιείται με την ειδική εντολή **return.**

Η λίστα παραμέτρων είναι στην ουσία, τα ορίσματα μιας μεθόδου, τα οποία συγκροτούν τις **υποκατάστατες παραμέτρους** αυτής, και πρέπει να είναι συμβατά από το σημείο που έγινε κλήση της μεθόδου με τις αντίστοιχες **πραγματικές** παραμέτρους.

Ένα καλό παράδειγμα προγράμματος είναι αυτό που ακολουθεί, το οποίο δέχεται σαν δεδομένα ένα κεφάλαιο που επενδύεται σε Repos για days (μέρες) με epitokio (επιτόκιο), όταν υπάρχει φόρος στον τόκο κατά foros (%). Μέσα στη μέθοδο Main ορίζεται το αντικείμενο invest της κλάσης Repos. Η κλήση της μεθόδου compute περνά με call by reference το invest και με call by value την τιμή της παραμέτρου foros. Η επιστροφή τιμής της compute δίνει το ποσό του φόρου που θα κατακρατήσει η εφορεία από τον τόκο, αλλά και το ποσό του τόκου (ακαθάριστος) που θα επιστρέψει μέσα από το αντικείμενο invest σαν μέλος invest.tokos. Η υποκατάστατη παράμετρος του invest είναι το αντικείμενο object μέσα στη μέθοδο compute της κλάσης Repos.

```
class Repos
{double kefalaio, epitokio, tokos;
 int days;
 double compute(Repos object, double foros)
    {
    object.tokos=(object.kefalaio*object.epitokio*object.days)/
    36500.0;
    return(foros*object.tokos)/100.0;//poso forou
    }
}

class java022
{public void static main(String x[])
{ Repos invest= new Repos();
  double eforia, foros ;
    try
    {
     invest.kefalaio=(new Double(x[0])).doubleValue();
     invest.epitokio=(new Double(x[1])).doubleValue();
     invest.days=(new Integer(x[2])).intValue();
     foros= (new Double(x[3])).doubleValue();
    if(invest.kefalaio<=0||invest.epitokio<=0||invest.days<=0||
       foros<=0)
     throw new NumberFormatException();
     eforia=invest.compute(invest,foros);
```

```
        }
        catch(ArrayIndexOutOfBoundsException e1)
        {
         System.out.println("Incomplete data…");
         return;
        }
        catch(NumberFormatException e2)
        { System.out.println("Errors in data…");
          return;
        }
        System.out.println("Tokos="+invest.tokos);
        System.out.println("Foros="+eforia);
        System.out.println("Clear Tokos="+(invest.tokos-eforia));
        System.out.println("Investment="+(invest.tokos-
            eforia+invest.kefalaio ));
    }
}
```

Το ισοδύναμο πρόγραμμα σε C# είναι το ακόλουθο:

```
using System;
class Repos
{ internal double kefalaio, epitokio, tokos;
  internal int days;
  internal virtual double compute(Repos object_Renamed, double
   foros)
    {
    object_Renamed.tokos=(object_Renamed.kefalaio*object_Rename
    d.epitokio * object_Renamed.days) / 36500.0;
    return (foros * object_Renamed.tokos) / 100.0;
    }
}
class java022
 {  static void Main(System.String[] x)
     { Repos invest=new Repos();
       double eforia, foros;
       try{ invest.kefalaio=(System.Double.Parse(x[0]));
            invest.epitokio=(System.Double.Parse(x[1]));
            invest.days=(System.Int32.Parse(x[2]));
            foros=(System.Double.Parse(x[3]));
if(invest.kefalaio<=0||invest.epitokio<=0||invest.days<=0||foro
    s<=0)
  { throw new System.FormatException();
  }
  eforia=invest.compute(invest,foros);
 }
  catch (System.IndexOutOfRangeException e1)
    {
        Console.WriteLine("incomplete data...");
```

```
            return ;
        }
        catch(System.FormatException e2)
        {
            Console.WriteLine("errors in data...");
            return ;
        }
    Console.WriteLine("tokos="+invest.tokos);
    Console.WriteLine("foros="+eforia);
    Console.WriteLine("clear tokos="+(invest.tokos-eforia));
    Console.WriteLine("investment="+(invest.tokos-
     eforia+invest.kefalaio));
  }
}
```

Ακολουθούν τρία χαρακτηριστικά αποτελέσματα εκτέλεσης του παραπάνω προγράμματος όπου τα δύο τελευταία συλλαμβάνουν την εξαίρεση:

Java022 400000 2.7 30 7
Tokos = 887.6712328767123
Foros=62.13698630136986
Clear tokos=825.534246573424
Investment=400825.5342465753

Java022 100000 3
Incomplete data...

Java022 –100000 3 s 4
Errors in data...

7.5 ΠΑΡΑΜΕΤΡΙΚΟΣ ΠΟΛΥΜΟΡΦΙΣΜΟΣ

Στη Java και στη C# είναι δυνατόν να υπάρχει μια μέθοδος με το ίδιο όνομα σε διάφορες αναπτύξεις και εκδόσεις μέσα στην ίδια την κλάση, αρκεί οι εκδόσεις αυτές της μεθόδου να διαφέρουν σε κάποιο στοιχείο στη λίστα παραμέτρων. Το φαινόμενο αυτό ονομάζεται **υπερφόρτωση μεθόδων (overloading methods)**, αλλά είναι και ένας βασικός τρόπος έκφρασης πολυμορφισμού της αντικειμενοστραφούς τεχνολογίας, γνωστός με το όνομα **παραμετρικός πολυμορφισμός(parametric polymorphism).** Οι ίδιες οι παρεχόμενες κλάσεις της Java είναι γεμάτες από υπερφορτωμένες μεθόδους. Για παράδειγμα στην κλάση **Math** η μέθοδος **abs** αντιπροσωπεύει μια γενική δράση, διότι εμφανίζεται εσωτερικά υπερφορτωμένη ισχύουσα έτσι για διάφορους τύπους δεδομένων.

Το θέμα στον παραμετρικό πολυμορφισμό είναι το πώς η Java και η C# αναγνωρίζουν σε ποια έκδοση της μεθόδου γίνεται η κλήση. Είναι λάθος να νομίζει κανείς πως παίζει κάποιο ρόλο η επιστρεφόμενη τιμή της μεθόδου στην επιλογή της έκδοσής της. Απλά ο κώδικας επιλέγει εκείνη που ταιριάζει περισσότερο στην λίστα παραμέτρων της κλήσης. Στον κώδικα που ακολουθεί φαίνεται καθαρά η έννοια του παραμετρικού πολυμορφισμού:

```
class polymorphism
{ void compute(int n, int m)
    { Console.WriteLine("version1➔"+(n+m));
    }
  void compute(int n, double m)
    { Console.WriteLine("version2➔"+(n+m));
    }
  void compute(double k)
    { Console.WriteLine("version3➔"+(k+k));
    }
  public static void Main(String[] x)
    { polymorphism object = new polymorphism();
      object.compute(5,9);
      object.compute(5,2.5);
      object.compute(2.5);
      object.compute(5);
    }
}
```

Το πρόγραμμα polymorphism είναι ένα παράδειγμα κλήσης της υπερφορτωμένης μεθόδου compute. Η έξοδος του προγράμματος είναι η ακόλουθη:

Polymorphism
Version 1➔ 14
Version 2➔ 7.5
Version 3➔ 5.0
Version 3➔ 10.0

Όπου στην τελευταία κλήση της μεθόδου compute, γίνεται αυτόματη επιλογή από τον κώδικα της C# η τρίτη έκδοση της μεθόδου.

Μέσα στα πλαίσια του πολυμορφισμού, η C# έχει τις **εικονικές (virtual) συναρτήσεις,** όπου μια συνάρτηση μπορούσε να εμφανίζεται σε διάφορες εκδόσεις σε υπερκλάση και υποκλάσεις. Και βέβαια είναι και αυτό ένα φαινόμενο υπερφόρτωσης μεθόδου, μόνο που τώρα ποια μέθοδος σε ποια υποκλάση (ή κλάση) θα κληθεί, προσδιορίζονται όχι στη διάρκεια του compilation, αλλά στη διάρκεια εκτέλεσης του προγράμματος. Η τεχνική κλήσης της μεθόδου είναι απλή, πρώτα ορίζεται μια **απλή αναφορά (reference)** στην υπερκλάση και ση συνέχεια αυτή ταυτίζεται με το αντικείμενο της υποκλάσης (ή κλάσης) που πρόκειται να γίνει κλήση της υπερφορτωμένης μεθόδου:

1. Ορισμός αναφοράς στην υπερκλάση
2. Ορισμός αντικειμένου στην υποκλάση(ή κλάση)
3. Εξίσωση (αντιστοίχηση) αναφοράς με το αντικείμενο
4. Κλήση της μεθόδου σαν μέλος της αναφοράς

Για παράδειγμα, η ακόλουθη κλάση υλοποιεί τρεις κλήσεις της υπερφορτωμένης μεθόδου common.

```
class reference
{
static void Main(String[] x)
    {
    Parent object_parent=new Parent();
    Son1 object_son1=new Son1();
    Son2 object_son2=new Son2();
    Parent ref;
    ref=object_parent;
    ref.common();
    ref=object_son1;
    ref.common();
    ref=object_son2;
    ref.common();
    }
}
```

Αυτό που στην ουσία συμβαίνει, είναι ότι στην πρώτη κλήση η αναφορά ref πήρε την κύρια μνήμη του αντικειμένου object_parent, ενώ στη συνέχεια πήρε σταδιακά τις μνήμες των object_son1 και object_son2.

7.6 CONSTRUCTORS

Μέσα στα πλαίσια των ειδικών μεθόδων μιας κλάσης, είναι και οι **κατασκευαστικές (constructors),** οι οποίες χρησιμοποιούνται για να αρχικοποιήσουν ένα αντικείμενο της κλάσης, θέτοντας τιμές στα μέλη, όπως είναι οι μεταβλητές της κλάσης. Οι αρχικές τιμές περνούν σαν παράμετροι τη στιγμή του ορισμού του αντικειμένου της κλάσης με βάση το συντακτικό τύπο ορισμού αντικειμένων:

[modifier] identifier([parameters])[:initializer([parameters])] {**body code**}

ένα απλό παράδειγμα για να γίνει κατανοητός ο παραπάνω τύπος είναι αυτό που ακολουθεί:

```
class MyClass:MySuperClass{
public MyClass(int x,string y):base(x,y){
constructor code}}
```

παρατηρούμε ότι το keyword της Java **Super** το οποίο μεταφέρει το έλεγχο στον constructor της υπερκλάσης, έχει αντικατασταθεί από το keyword **base**. Οι τιμές που παίρνει ο Modifier φαίνονται στον παρακάτω πίνακα:

	Member of class	Member of struct
Public	√	√
Protected	√	N/A
Private	√	√
Internal	√	√
protected internal	√	N/A

New	N/A	N/A
Abstract	N/A	N/A
Sealed	N/A	N/A
Virtual	N/A	N/A
Override	N/A	N/A
Readonly	N/A	N/A
Volatile	N/A	N/A
Static	√	√
Extern	√	√

Ωστόσο, οι constructors, σαν ειδικές μέθοδοι υπακούουν στους ακόλουθους ειδικούς κανόνες:

- **Έχουν το ίδιο όνομα με την κλάση**
- **Δεν επιστρέφουν τιμή και δεν είναι void**
- **Η κλήση τους γίνεται αυτόματα και μόνο με τον ορισμό του αντικειμένου**
- **Επιτρέπεται η υπερφόρτωσή τους που είναι και το σύνηθες φαινόμενο**
- **Η βασική τους έκδοση διευθετεί κάθε μεταβλητή της κλάσης τους**

Στο πρόγραμμα java028 που ακολουθεί, η κλάση Squares χρησιμοποιεί δυο μεταβλητές τις platos και mikos, την κανονική μέθοδο Square() που επιστρέφει το εμβαδόν του ορθογωνίου με διαστάσεις τις δυο μεταβλητές της κλάσης και τρεις εκδόσεις υπερφορτωμένης κατασκευαστικής μεθόδου. Η πρώτη είναι βασική, η δεύτερη με κενή λίστα παραμέτρων και η τρίτη για την ειδική περίπτωση που το ορθογώνιο είναι τετράγωνο.

```
class Squares
 { double platos;
   double mikos;
   Squares(double p, double m)
    { platos=p;
      mikos=m;
    }
   Squares()
    { platos=0;
      mikos=0;
    }
    Squares(double dim)
     { platos=mikos=dim;
     }
    double square()
     { return platos*mikos;
     }
 }
class java028
 { public static void main(String args[])
   { Squares shape1=new Squares();
     Squares shape2=new Squares(30,20);
```

```
        Squares shape3=new Squares(10);
        System.out.println("Square of Shape1="+shape1.square());
        System.out.println("Square of Shape2="+shape2.square());
        System.out.println("Square of Shape3="+shape3.square());
      }
   }
```

Το ισοδύναμο πρόγραμμα σε C# είναι το ακόλουθο

```
using System;
class Squares
 {  internal double platos;
    internal double mikos;
    internal Squares(double p, double m)
     { platos = p;
       mikos = m;
     }
    internal Squares()
     { platos = 0;
       mikos = 0;
     }
    internal Squares(double dim)
     { platos = mikos = dim;
     }
    internal virtual double square()
     { return platos * mikos;
     }
}
class java028
 {
    [STAThread]
    public static void  Main(System.String[] args)
    { Squares shape1 = new Squares();
      Squares shape2 = new Squares(30, 20);
      Squares shape3 = new Squares(10);
      Console.WriteLine("Square of Shape1="+shape1.square());
      Console.WriteLine("Square of Shape2="+shape2.square());
      Console.WriteLine("Square of Shape3="+shape3.square());
     }
}
```

Η εκτέλεση του προγράμματος θα δώσει την ακόλουθη έξοδο:

Square of Shape1= 0.0
Square of Shape2= 600.0
Square of Shape3= 100.0

Στην κλάση java028 ορίζονται τρία αντικείμενα shape1, shape2, shape3 της κλάσης Squares. Το πρώτο, τη στιγμή του ορισμού του, βέβαια, προκαλεί την έκδοση της κατασκευαστικής μεθόδου της Squares με κενή λίστα παραμέτρων.

Το δεύτερο αντικείμενο, ενεργοποιεί τη βασική έκδοση της κατασκευαστικής, ενώ το τρίτο, την τρίτη έκδοση όπου το platos και το mikos είναι ίσα.

Η κατασκευαστική μέθοδος δεν είναι υποχρεωτική για μια κλάση. Όταν δεν υπάρχει, τότε αναγκαστικά τα οριζόμενα αντικείμενα της κλάσης πρέπει να δίδονται με κενή λίστα παραμέτρων. Στη Java ο ορισμός κάθε κλάσης εμπεριέχει αυτόματα την ειδική μέθοδο **finalize**() η οποία καλείται αυτόματα από το διερμηνέα της Java με σκοπό την αποκομιδή σκουπιδιών (**garbage collection).** Στην C# η μέθοδος finalize της Java, αντικαθιστάται από ένα destructor, ο οποίος είναι μια μέθοδος με όνομα ίδιο με αυτό της κλάσης και προσθέτοντας μπροστά το σύμβολο **(~)**.

7.7 ΔΗΛΩΣΗ ΚΛΑΣΗΣ

Μέχρι τώρα, έχει γίνει βέβαια ευρεία χρήση της κλάσης, χωρίς όμως να έχει δοθεί ο πλήρης ορισμός της με τον πλήρη συντακτικό της τύπο. Η κλάση είναι το βασικό κύτταρο του αντικειμενοστραφούς προγραμματισμού και καλύπτει πλήρως την έννοια του προγραμματιστικού ολονίου. Η **κλάση (class)** ορίζεται με τον ακόλουθο γενικό συντακτικό τύπο:

[κατηγορία] class όνομα_κλάσης[: όνομα_υπερκλάσης]
{δήλωση μεταβλητών κλάσης;
Μέθοδοι κλάσης;
Εκδόσεις κατασκευαστικής μεθόδου; }

Ισχύουν τα ακόλουθα:
Κατηγορία: προσδιορίζει τον μετατροπέα της κλάσης ο οποίος μπορεί να πάρει μια από τις τιμές που έχουν ήδη αναλυθεί:

- **Public**
- **Final**
- **sealed**
- **abstract**

7.8 ΑΣΚΗΣΕΙΣ

1. Έστω ο παρακάτω κώδικας:

```
Class finalclass
{ final int finalvar;
  public void setfinalvar()
   { finalvar=1;
   }
}
```

Να γράφει ο κώδικας σε C# και να βρεθεί το λάθος.

2. Έστω ο παρακάτω κώδικας:

```
Class zero
{ static int a=5,y;
  static void write(int b)
   { System.out.println("b="+b+"a="+a+"y="+y);
   }
```

```
  static{y=a++;}
  public static void main(String s[])
   { write(10);
   }
 }
```

Να γράφει ο κώδικας σε C#, να δοθεί η έξοδος του προγράμματος και να δικαιολογηθεί πλήρως.

3. Μέσα σε ένα ορθογώνιο οικόπεδο υπάρχουν ένα ορθογώνιο κτίσμα και δυο τετράγωνες αποθήκες. Να γραφεί πρόγραμμα C# υπολογισμού του κήπου (ακάλυπτου χώρου) του οικοπέδου αυτού. Όλες οι διαστάσεις αποθηκών και κτίσματος καθώς και του αρχικού οικοπέδου δίνονται σαν ορίσματα στη γραμμή εκτέλεσης του προγράμματος. Το πρόγραμμα προβλέπει κάθε πιθανό σημείο κατάρρευσης.

ΚΕΦΑΛΑΙΟ 8ο

8. ΠΑΡΕΧΟΜΕΝΕΣ ΚΛΑΣΕΙΣ-ΜΕΘΟΔΟΙ

8.1 Η ΕΝΝΟΙΑ ΤΟΥ ΠΑΚΕΤΟΥ

Το **πακέτο (package)** έχει πολλή μεγαλύτερη ηλικία απ' ότι ο νέος μηχανικός λογισμικού μπορεί να φανταστεί. Ήδη το 1983 υπήρχε σαν επίσημη γλώσσα προγραμματισμού του DOD(Department Of Defense) των ΗΠΑ, η **ANSI Standard ADA.** Αυτή, λοιπόν, η γλώσσα προγραμματισμού είχε από παλιά έννοιες που μόλις σήμερα σύγχρονα εργαλεία, όπως η Java και η C# προσπαθούν να αγγίξουν. Αντίθετα οι άλλες γλώσσες προγραμματισμού (FORTRAN, BASIC, COBOL) προσπαθούσαν να οργανώσουν τις **μονάδες μεταγλώττισης** (**compilation units**) μέσα από στατικές ή δυναμικές βιβλιοθήκες (*.lib ή *.dll), η ADA εισήγαγε σαν μέσο οργάνωσης των μονάδων της την έννοια του πακέτου.

Οι τρόποι οργάνωσης στα **programming holons** (συναρτήσεις, υποπρογράμματα, μέθοδοι, κλάσεις) στην Τεχνολογία Λογισμικού, ακολουθούν σε κάθε γλώσσα προγραμματισμού ή εργαλείο ανάπτυξης λογισμικού, μια από τα ακόλουθες μεθόδους:

1. Βιβλιοθήκες (libraries): ο προγραμματιστής δημιουργεί βιβλιοθήκες, οι οποίες περιλαμβάνουν αρχεία της μορφής *.obj. Ο κώδικας των μονάδων αυτών ενσωματώνεται στο τελικό *.exe πρόγραμμα από τον linker του όλου συστήματος. Αν η ενσωμάτωση γίνεται ολοκληρωτικά στο linking της εφαρμογής, τότε το παραγόμενο *.exe είναι τελικής μορφής και τρέχει αυτοδύναμα, ανεξάρτητα από τις βιβλιοθήκες που χρησιμοποίησε. Αυτό το είδος βιβλιοθηκών ονομάζονται στατικές βιβλιοθήκες, και έχουν ονόματα της μορφής *.lib. Αν όμως το εκτελέσιμο πρόγραμμα ενσωματώνει σταδιακά στη διάρκεια της εκτέλεσης τον κώδικα των μονάδων του, από βιβλιοθήκες δυναμικές του τύπου *.dll, τότε οι χρησιμοποιούμενες βιβλιοθήκες θα πρέπει να είναι ορατές στον εκτελέσιμο κώδικα. Σήμερα, ο τρόπος οργάνωσης των προγραμματιστικών μονάδων σε βιβλιοθήκες, θεωρείται ξεπερασμένος, και δύσκολος στη διαχρονική ζωή της εφαρμογής.

2. Πακέτα (packages): στην περίπτωση του **OOP (OBJECT ORIENTED PROGRAMMING),** τα προγραμματιστικά ολόνια συγκροτούνται από κλάσεις, οι οποίες οργανώνονται σε πακέτα. Κατά κανόνα οι προγραμματιστικές μονάδες που ανήκουν στο ίδιο πακέτο, ικανοποιούν κάποιο δείκτη ομοιότητας με την κλασσική έννοια της **Ταξονομίας (Taxonomy)**. Για παράδειγμα το πακέτο της Java για είσοδο/ έξοδο, ονομάζεται java.io, το οποίο συγκροτείται από κλάσεις που μοιάζουν μεταξύ τους, διότι ασχολούνται όλες με διευθέτηση δεδομένων από διάφορα μέσα. Τα πακέτα μπορούν αν είναι παρεχόμενα από τη γλώσσα προγραμματισμού ή/ και πακέτα που έχει ο προγραμματιστής κατασκευάσει. Ωστόσο, σε κάθε περίπτωση, ο χρήστης μπορεί να εμπλουτίσει αυτά με δικές του κλάσεις και προγραμματιστικές μονάδες γενικότερα. Η έννοια του πακέτου θεωρείται σήμερα σαν μια ανοικτής αρχιτεκτονικής μέθοδος οργάνωσης των

στοιχείων εφαρμογής και είναι στην πρώτη γραμμή σε διεθνές επίπεδο, με προοπτική να παραμείνει για αρκετά χρόνια.

Η μεταβλητή περιβάλλοντος (environmental variable) **CLASSPATH** είναι πολλές φορές απαραίτητη για την ορατότητα των αρχείων *.class κάποιου πακέτου. Αν υπάρχουν προβλήματα στο θέμα αυτό, τότε ο χρήστης πρέπει να ενσωματώσει την πληροφορία της μεταβλητής αυτής τον αρχικό κατάλογο της ιεραρχίας των κλάσεων που εργάζεται.

Παρόλο που η Java και η C# παρουσιάζουν μεγάλες ομοιότητες στη χρήση και στις λέξεις κλειδιά των πακέτων, ωστόσο, εξ' αρχής τονίζεται ότι όταν η Java εισάγει (πρόκειται να χρησιμοποιήσει κώδικα) απο κάποιο πακέτο, χρησιμοποιεί τη λέξη-κλειδί **import**. Η C# χρησιμοποιεί την αντίστοιχη λέξη-κλειδί **using**. Τέλος, η λέξη-κλειδί **package** στη Java αντιστοιχεί στη C# με τη λέξη-κλειδί **namespace.**

8.2 ΟΡΙΣΜΟΣ ΠΑΚΕΤΟΥ

Ένα πηγαίο πρόγραμμα σε Java, επιτρέπει στον προγραμματιστή, να δημιουργεί ένα δικό του πακέτο να χρησιμοποιεί ένα πακέτο που υπάρχει και διατίθεται από την ίδια τη γλώσσα ή/ και να χρησιμοποιεί ένα δικό του πακέτο που ήδη υπάρχει και έχει δημιουργηθεί.

Ο χρήστης για να δημιουργήσει ένα πακέτο, πρέπει σαν πρώτη και καλύτερη εντολή να υπάρχει η εντολή δήλωσης με συντακτικό τύπο:

package p1[. p2[….pm]]…];

όπου το όνομα του πακέτου ταυτίζεται με τη χωροθέτησή του σαν αρχείο μέσα στο δένδρο καταλόγων του δίσκου. Αυτό σημαίνει πως το πακέτο θα βρίσκεται στο μονοπάτι: p1/p2/…/pm.

Για παράδειγμα, αν δοθεί η εντολή: package jcp.one.two; τότε στο κατάλογο jcp υπάρχει ο υποκατάλογος one ο οποίος έχει υποκατάλογο τον υποκατάλογο two που περιέχει τον πηγαίο του πακέτου αυτού.

Στο σημείο αυτό, πρέπει να τονιστεί ιδιαίτερα πως κάθε αρχείο Java που μπορεί να υποστεί μεταγλώττιση, **είναι αναγκαστικά τμήμα κάποιου πακέτου.** Απλούστατα, αν δεν υπάρχει η εντολή δήλωσης package, τότε στο αντίστοιχο αρχείο *.class τοποθετείται αυτόματα μια αντίστοιχη δήλωση όπου το πηγαίο τοποθετείται σε ένα **γενικό ανώνυμο πακέτο.** Στη συνέχεια ακολουθεί ένα παράδειγμα δημιουργίας πακέτου σε Java.

```
package jcp;
class Telephones
{ String name;
  String homenumber;
  String mobile;
  Telephones(String str1, String str2, String str3)
   { name=str1;
     homenumer=str2;
     mobile=str3;
```

```
 }
  void show_list()
   { System.out.println(name+” ”+homenumber+” “ +mobile);
   }
}
class Java029
{ public static void main( String x[])
  { Telephones Record[]= new Telephones[4];
    Record[0]= new Telephones(“name1”,
   “2101111111”,”6941111111”);
    Record[1]= new Telephones(“name2”,
   “2102222222”,”6942222222”);
    Record[2]= new Telephones(“name3”,
   “2103333333”,”6943333333”);
    Record[3]= new Telephones(“name4”,
   “2104444444”,”6944444444”);
   for (int i=0;i<4;i++)
     Record[i].show_list();
  }
}
```

Στη C# ο όρος package, όπως είδαμε και παραπάνω, αντιστοιχεί στη λέξη namespace. Εσωτερικά του namespace, περιλαμβάνουμε κλάσεις που χρησιμοποιούμε στο ίδιο project. Η C# έχει τη δυνατότητα να υποστηρίζει πολλαπλά namespaces σε ένα συγκεκριμένο φάκελο, εξαλείφοντας τις απαιτήσεις της Java για ιεραρχίες πακέτων έτσι ώστε να χαρτογραφεί τη δομή του συστήματος αρχείων της. Για την καλύτερη κατανόηση των παραπάνω, μετατρέπουμε το πρόγραμμα Java029 ως εξής:

```
namespace jcp
{  using System;
   class Telephones
    {  internal System.String name;
       internal System.String homenumber;
       internal System.String mobile;
       internal Telephones(System.String str1,System.String
       str2, System.String str3)
        { name = str1;
          homenumber = str2;
          mobile = str3;
        }
     internal virtual void  show_list()
      {
        System.Console.WriteLine(name +"" + homenumber + ""+
        mobile);
      }
   }
    class Java029
     {
      [STAThread]
```

```
public static void  Main(System.String[] x){
   Telephones[] Record = new Telephones[4];
  Record[0]= new Telephones("name1",
        "2101111111","6941111111");
  Record[1]= new Telephones("name2",
        "2102222222","6942222222");
  Record[2]= new Telephones("name3",
        "2103333333","6943333333");
  Record[3]= new Telephones("name4",
        "2104444444","6944444444");
   for (int i = 0; i < 4; i++)
     Record[i].show_list();}}}
```

8.3 ΧΡΗΣΗ ΣΥΣΤΑΤΙΚΩΝ ΠΑΚΕΤΩΝ

Μέχρι τώρα, αναλύθηκε ο τρόπος δημιουργίας πακέτου, όπου το εκτελέσιμο πρόγραμμα, ανήκε μέσα στο ίδιο πακέτο. Ωστόσο, αυτό που έχει αξία, είναι η δημιουργία του πακέτου να είναι προγραμματιστικά ανεξάρτητη δραστηριότητα από τη χρήση των συστατικών του πακέτου από διάφορα άλλα προγράμματα εκτός αυτού. Αυτό σημαίνει, πως μας ενδιαφέρει του πακέτο να μοιάζει σαν την έννοια της **βιβλιοθήκης (library)** στον παλαιομοδίτικο προγραμματισμό, όπου τα προγράμματα κάνουν χρήση των object συστατικών της βιβλιοθήκης με τη βοήθεια linker, ενσωματώνοντας τον αντίστοιχο κώδικα στο *.exe ή θέτοντας τον εκτελέσιμο κώδικα σε ετοιμότητα ενσωμάτωσης στη διάρκεια της εκτέλεσης του προγράμματος στην περίπτωση της **δυναμικής βιβλιοθήκης** *.dll.

Τόσο στη Java όσο και στη C# το πακέτο έχει αντικαταστήσει την έννοια της βιβλιοθήκης. Η χρήση συστατικών ενός πακέτου, γίνεται με τη βοήθεια του ονόματος αυτού. Μπορεί να γίνει με έναν από τους ακόλουθους δύο τρόπους:

1. Με ορισμό αντικειμένου σε κλάση του πακέτου με τη βοήθεια του ονόματος του πακέτου **άμεσα**, την ίδια εντολή του ορισμού. Για παράδειγμα, στη Java αυτό δηλώνεται ως εξής:

   java.util.Stack= new java.util.Stack();

   ενώ αντίστοιχα στη C#:

   System.Collections.Stack= new System.Collections.Stack();

   Ορίζει το αντικείμενο object που χρησιμοποιεί την κλάση Stack του πακέτου. Ωστόσο, αυτός ο άμεσος τρόπος κλήσης συστατικών πακέτων, δεν είναι και ο πλέον κατάλληλος, ιδιαίτερα όταν έχουμε μεγάλα μακρόσυρτα ονόματα στα πακέτα μας.

2. Με την **εισαγωγή** μέρους ή και όλου, των συστατικών πακέτου στη αρχή του αρχείου με την εντολή δήλωσης:

   import p1.[.p2].(όνομα κλάσης/*);

   και αντίστοιχα στη C# με την εντολή δήλωσης:

using p1.[.p2].(όνομα κλάσης);

Όπου θα πρέπει να δοθεί το όνομα της κλάσης που θα χρησιμοποιηθεί. Είναι λοιπόν φανερό, πως η χρήσης της ειδικής εντολής **using** συνίσταται για χρήση συστατικών πακέτων.

Η τελευταία ερώτηση που απαιτεί μια ξεκάθαρη απάντηση, είναι για τον τρόπο δήλωσης των συστατικών πακέτου που χρησιμοποιείται από ανεξάρτητα από αυτό προγράμματα (σαν βιβλιοθήκη). Η απάντηση είναι απλή. **θα πρέπει όλα τα συστατικά (κλάσεις και μέθοδοι) να δηλωθούν σαν public**, για να είναι δυνατή η χρήση τους εκτός του πακέτου τους, χωρίς να απαιτείται βέβαια αυτές οι κλάσεις να είναι υποκλάσεις των κλάσεων του πακέτου.

Υπάρχουν περιπτώσεις που πρέπει να δηλωθούν αντικείμενα που ορίζονται πάνω σε **κλάσεις συνώνυμες** που ανήκουν βέβαια σε διαφορετικά πακέτα. Σε τέτοιες περιπτώσεις, οι κλάσεις αυτές δεν είναι δυνατόν να προσπελαθούν απλά με το όνομά τους και τη χρήση της ειδικής εντολής **using,** αλλά απαιτείται η χρήση του πλήρους ονόματος, δηλαδή πρέπει να τηρηθεί ο πρώτος, άμεσος τρόπος δήλωσης των αντικειμένων που θα οριστούν για παράδειγμα οι ακόλουθες εντολές:

```
System.MyLib.Cl1 object1= new System.MyLib.Cl1();
System.YourLib.Cl1 object2= new System.YourLib.Cl1();
```

ορίζουν τα αντικείμενα object1, object2 σε συνώνυμη κλάση Cl1, δύο διαφορετικών πακέτων System.MyLib, System.YourLib.

8.4 ΠΑΡΑΔΕΙΓΜΑ ΠΑΚΕΤΟΥ

Ήδη αναλύθηκε πως είναι δυνατόν να οριστεί ένα πακέτο, με κύριο στόχο τα συστατικά του να μπορούν να προσπελασθούν από κλάσεις άλλων πακέτων. Ειδικά στην περίπτωση που το πακέτο δεν έχει κλάση που να περιέχει την ειδική μέθοδο **Main()**, τότε μιλάμε για **πακέτο-βιβλιοθήκη,** το οποίο έχει σαν στόχο να δανείζει τις κλάσεις του σε άλλα προγράμματα. Σε ένα τέτοιο πακέτο ανακεφαλαιώνοντας, ο αναγνώστης πρέπει να έχει υπ' όψιν του τους ακόλουθους **κανόνες,** οι οποίοι έχουν ήδη αναλυθεί στο μεγαλύτερο μέρος τους:

1. Τα συστατικά του πακέτου πρέπει να είναι **public**
2. Τα αντικείμενα που δηλώνονται πάνω σε κλάσεις του πακέτου, δηλώνονται με τη χρήση της δήλωσης using
3. Η πρώτη εντολή του πακέτου είναι αναγκαστικά η δήλωση namespace.
4. Στο πακέτο-βιβλιοθήκη δεν έχει νόημα η ενσωμάτωση της μεθόδου Main().
5. Η εκτέλεση των προγραμμάτων C# και Java αντίστοιχα που χρησιμοποιούν ένα πακέτο γίνεται κανονικά από εκεί που βρίσκεται το βασικό ενδιάμεσο αρχείο εκτέλεσης που ενσωματώνει τη Main(). Αν η εκτέλεση γίνει ένα επίπεδο πάνω από το πακέτο που το χρησιμοποιεί, κατά κανόνα δεν υπάρχει πρόβλημα εντοπισμού των ενδιάμεσων αρχείων. Διαφορετικά γίνεται χρήση της ειδικής μεταβλητής περιβάλλοντος classpath στη Java για τους υποκαταλόγους έρευνας - εντοπισμού των ενδιάμεσων πακέτων

6. Στο Διαδίκτυο, κατά κανόνα τα πακέτα εμφανίζονται από πλευράς οργάνωσης στο δίσκο, κατά την αντίστροφη έννοια του πεδίου μέσα στο Internet. Για παράδειγμα, αν το πεδίο του Διαδικτύου είναι το a.b.com τότε τα πακέτα μας θα πρέπει να τηρούν τη διάταξη com.b.a.

Σαν παράδειγμα πακέτου, ας θεωρήσουμε τον κατάλογο jcp όπου υπάρχει το πακέτο jcp. Αυτό περιέχει την κλάση Telephone και τη μέθοδο Show_List() η οποία απλά εμφανίζει στη οθόνη ένα όνομα με τα τηλέφωνα κατοικίας και κινητού. Τα πάντα έχουν δηλωθεί public, ακόμη και η βασική κατασκευαστική μέθοδος της κλάσης Telephones.

```
package jcp;
public class Telephones{
String name;
String homenumber;
String mobile;

public Telephones(String str1, String str2, String str3){
name=str1;
homenumber=str2;
mobile=str3;
}
public void show_list(){
System.out.println(name+" "+homenumber+" " +mobile);}}
```

Στη συνέχεια, μέσα στον κατάλογο jcp, δίδεται το πρόγραμμα Java030 το οποίο χρησιμοποιεί το πακέτο jcp με τη βοήθεια της δήλωσης import

```
import jcp.*;
class Java030
{ public static void main( String x[])
   {
    Telephones Record[]= new Telephones[4];
    Record[0]= new Telephones("name1",
   "2101111111","6941111111");
    Record[1]= new Telephones("name2",
   "2102222222","6942222222");
    Record[2]= new Telephones("name3",
   "2103333333","6943333333");
    Record[3]= new Telephones("name4",
   "2104444444","6944444444");
    for (int i=0;i<4;i++)
      Record[i].show_list();
   }
}
```

Το αντίστοιχο πρόγραμμα σε C# έχει την παρακάτω ισοδύναμη μορφή

```
namespace jcp
{ using System;
```

```
  public class Telephones
   { System.String name;
     System.String homenumber;
     System.String mobile;
     public Telephones(System.String str1,System.String str2,
       System.String str3)
       { name = str1;
         homenumber = str2;
          mobile = str3;
        }
    public void  show_list()
     { System.Console.WriteLine(name +"" + homenumber + ""+
       mobile);
     }
}
```

Στη συνέχεια ο χρήστης θα τρέξει το πρόγραμμα Java030, όπου θα πρέπει να καλέσει με τη βοήθεια της δήλωσης using εκτός από τα πακέτα της κλάσης System και το πακέτο jcp.

```
namespace Java030
 { using System;
   using jcp;
   class Java030
    {
    [STAThread]
     public static void  Main(System.String[] x)
       { Telephones[] Record = new Telephones[4];
         Record[0]= new Telephones("name1",
             "2101111111","6941111111");
         Record[1]= new Telephones("name2",
             "2102222222","6942222222");
         Record[2]= new Telephones("name3",
             "2103333333","6943333333");
         Record[3]= new Telephones("name4",
             "2104444444","6944444444");
        for (int i = 0; i < 4; i++)
          Record[i].show_list();
       }
    }
}
```

Τρέχοντας ο χρήστης το πρόγραμμα Java031, το οποίο αποτελεί μια παραλλαγή του τελευταίου προγράμματος, κάνει μια απ' ευθείας χρήση του πακέτου jcp όπως φαίνεται παρακάτω.

```
namespace Java031
 { using System;
   class Java031
    {
```

```
[STAThread]
 public static void  Main(System.String[] x)
  { jcp.Telephones[] Record = new jcp.Telephones[4];
    Record[0]= new jcp.Telephones("name1",
        "2101111111","6941111111");
    Record[1]= new jcp.Telephones("name2",
        "2102222222","6942222222");
    Record[2]= new jcp.Telephones("name3",
        "2103333333","6943333333");
    Record[3]= new jcp.Telephones("name4",
        "2104444444","6944444444");
    for (int i = 0; i < 4; i++)
      Record[i].show_list();
  }
 }
}
```

8.5 ΟΡΙΣΜΟΣ ΔΙΑΠΡΟΣΩΠΕΙΑΣ

Οι γνώστες της C++ γνωρίζουν ήδη την έννοια της πολλαπλής κληρονομικότητας. Στη Java όπως και σε όλο το Διαδικτυακό προγραμματισμό ισχύει, η **απλή κληρονομικότητα (single inheritance).** Αυτό σημαίνει ότι μια υποκλάση προέρχεται αυστηρά ιεραρχικά από μια ακριβώς κλάση. Ωστόσο, οι σχεδιαστές της Java και της C#, προσπάθησαν να υπερβούν την αδυναμία αυτή με το φαινόμενο του **πολυμορφισμού (polymorphism)**, χρησιμοποιώντας σαν όχημα μηχανισμού την έννοια της **διαπροσωπείας (interface),** όπου δίνεται το τι μπορεί να γίνει αλλά όχι και ο τρόπος υλοποίησης, ο οποίος μπορεί να είναι πολλαπλός και να υπάρχει σε διάφορες άλλες κλάσεις της εφαρμογής μας.

Είναι λοιπόν φανερό, πως μια διαπροσωπεία, θυμίζει έντονα την έννοια της αφηρημένης κλάσης και εν μέρει αυτό είναι ορθό. Η διαπροσωπεία είναι μια **πλήρη αφηρημένη κλάση (complete abstract class)** με μεθόδους αφηρημένες. Οι διαπροσωπείες, όπως και οι κλάσεις, ορίζουν ιεραρχίες κληρονομικότητας με δυνατότητα δήλωσης μεταβλητών με τύπο αναφοράς σε αυτές. Εδώ η βασική ιδέα, είναι άλλες υποκλάσεις να επεκτείνουν τις διαπροσωπείες, δηλαδή στην ουσία να περιέχουν τα σώματα των εντολών των μεθόδων των διαπροσωπειών. Ο γενικός συντακτικός τύπος ορισμού διαπροσωπείας έχει την ακόλουθη δομή:

```
Ορατότητα interface όνομα_ interface[:interface που επεκτείνει]
                {
                τύπος τιμής μέθοδος1(λίστα παραμέτρων);
                                .....

                τύπος τιμής μέθοδοςm(λίστα παραμέτρων);
                }
```

όπου, αν η **ορατότητα** έχει τιμή **public,** τότε η διαπροσωπεία μπορεί να χρησιμοποιηθεί και από άλλα πακέτα, διαφορετικά είναι δυνατή η χρήση της μόνο μέσα στα πλαίσια του ίδιου πακέτου. **Αν η διαπροσωπεία έχει δηλωθεί σαν public, τότε κάθε συστατικό της είναι αυτόματα επίσης public χωρίς**

αντίστοιχα τέτοια δήλωση. Αν υπάρχουν μεταβλητές στη διαπροσωπεία, θεωρούνται αυτόματα εσωτερικά (implicitly) σταθερές με αναγκαστική, βέβαια την αρχική (και τελική στην ουσία τιμή τους). Σώματα εντολών των μεθόδων της διαπροσωπείας απαγορεύεται να υπάρχουν. Ένα απλό παράδειγμα διαπροσωπείας με δυο μεθόδους είναι το ακόλουθο, όπου η πρώτη μέθοδος παίρνει με call by value μια ακέραια παράμετρο και η δεύτερη ταυτίζεται με την έννοια του υποπρογράμματος:

```
interface Inter1
{ int method1(int param1);
  void method2();
}
```

Στο ακόλουθο παράδειγμα η κλάση Example1 υλοποιεί τις μεθόδους της διαπροσωπείας Inter1, ενώ παράλληλα χρησιμοποιεί δικές της μεθόδους και μεταβλητές:

```
class Example1 implements Inter1{
double var1;
int var2;
public int method1(int m){
if(m<=0)
m=m*m*m;
return m;
}
public void method2(){
System.out.println("I love Java");
}
void local method(){
System.out.println("I am just a member");
}}
```

Το ισοδύναμο πρόγραμμα σε C# φαίνεται στον κώδικα που ακολουθεί. Παρατηρείστε ότι έχει ελάχιστες διαφορές σε σχέση με το παραπάνω πρόγραμμα Java

```
interface Inter1
 { int method1(int param1);
   void method2();
 }
using System;
namespace Example1
 { class Example1: Inter1
    { double var1;
      int var2;
       public int method1(int m)
        { if(m<=0) m=m*m*m;
           return m;
         }
```

```
        public void method2()
          { Console.WriteLine("I love C#");
          }
        void local method()
          { Console.WriteLine("I am just a member");
      }
    }
}
```

8.6 ΧΡΗΣΗ ΔΙΑΠΡΟΣΩΠΕΙΑΣ

Χρειάζεται προσοχή στον τρόπο κλήσης μεθόδων που ανήκουν στη διαπροσωπεία. Ο κώδικας που θα εκλεγεί από την κλήση, θα έρθει δυναμικά, μέσα από απλούς κανόνες ιεράρχησης της κληρονομικότητας. Υπάρχει γενικός συντακτικός τύπος ορισμού αντικειμένου για διαπροσωπείες μέσα στα προγράμματά μας:

int_name object = new cl_name();

όπου **int_name** το όνομα του interface, του οποίου της μεθόδους αναπτύσσει η κλάση **cl_name**. Το οριζόμενο αντικείμενο **object** είναι σε σχέση με τη διαπροσωπεία και επομένως δεν μπορεί να αντιπροσωπεύει μέλη της κλάσης cl_name που δεν είναι παράλληλα και μέλη του interface int_name.

Όλα τα παραπάνω γίνονται κατανοητά στο επόμενο παράδειγμα που αποτελεί επέκταση του παραδείγματος της προηγούμενης παραγράφου:

```
interface Inter1
{ int method1(int param1);
  void method2();
}
using System;
namespace Example1
 {  class Example1: Inter1
    {  double var1;
       int var2;
       public int method1(int m)
        { if(m<=0) m=m*m*m;
           return m;
        }
       public void method2()
         { Console.WriteLine("I love C#");
         }
       void local method()
        { Console.WriteLine("I am just a member");
        }
        static void Main(string[] args)
         { Inter1 object = new Example1();
          object.method2();
          Console.WriteLine("Value="+ object.method1(2));
```

```
        }
    }
}
```

θα έχουμε μια πλήρη έξοδο της μορφής:

```
I love C#
Value=8
```

Παρατηρούμε πως το αντικείμενο object έχει νόημα να αναφέρεται στα μέλη της διαπροσωπείας Inter1, όπως είναι οι μέθοδοι method1, method2, ενώ θα ήταν εντελώς λάθος να εμφανιστεί μέσα στο πρόγραμμα μια κλήση της μορφής:

```
object.localmethod();
```

8.7 ΜΕΡΙΚΗ ΑΝΑΠΤΥΞΗ & ΕΠΕΚΤΑΣΗ ΔΙΑΠΡΟΣΩΠΕΙΑΣ

Όπως έχει ήδη αναφερθεί, μια κλάση που αναφέρεται σε διαπροσωπεία θα πρέπει να υλοποιεί όλες τις μεθόδους αυτής. Ωστόσο, αυτό δεν είναι απόλυτα ακριβές. Τόσο η Java, όσο και η C# δίνουν τη δυνατότητα της **μερικής ανάπτυξης μεθόδων διαπροσωπείας, αρκεί η αντίστοιχη κλάση που υλοποιεί αυτό, να δηλωθεί σαν αφηρημένη (abstract).** Βέβαια, σε μια τέτοια ειδική περίπτωση, κάθε υποκλάση μιας τέτοιας κλάσης, θα πρέπει να υλοποιεί τις μεθόδους της διαπροσωπείας ή θα πρέπει επίσης να είναι **abstract**. Για παράδειγμα, η ακόλουθη κλάση κάνει μερική ανάπτυξη του interface Inter1 της παραγράφου 8.5:

```
abstract class Partial: Inter1{
double g,f;
void display(){
System.Console.WriteLine("g+f"+ (g+f));}
public void method2(){
System.Console.WriteLine("I love C#…");}}
```

Η κλάση Partial πρέπει να δηλωθεί σαν abstract, διότι κάνει μερική υλοποίηση των μεθόδων του interface Inter1. παρατηρούμε πως, ναι μεν υλοποιείται η μέθοδος method2, αλλά απουσιάζει η υλοποίηση της μεθόδου method1 του Inter1.

Η διαπροσωπεία δεν παύει να είναι σαν έννοια, μια κλάση ειδικής μορφής. Επομένως, είναι δυνατή **η επέκταση** μιας διαπροσωπείας. Για παράδειγμα, η διαπροσωπεία του ToInter1 είναι μια επέκταση της γνωστής μας Inter1:

```
interface ToInter1:Inter1{
void method3();}
```

προσοχή, όμως, σε μια τέτοια περίπτωση θα πρέπει μια κλάση που υλοποιεί τις μεθόδους του ToInter1 να τις υλοποιεί όλες (method1, method2, method3), εκτός, βέβαια της μερικής ανάπτυξης που ήδη αναλύθηκε.

8.8 ΔΙΑΠΡΟΣΩΠΕΙΕΣ ΚΑΙ ΠΟΛΥΜΟΡΦΙΣΜΟΣ

Όπως ήδη έχει αναφερθεί, οι διαπροσωπείες, καλύπτουν την ανάγκη για **πολλαπλή κληρονομικότητα,** δηλαδή την επέκταση περισσότερων της μιας κλάσης, το οποίο, βέβαια δεν υποστηρίζεται κανονικά από τον Διαδικτυακό προγραμματισμό. Επίσης, ενώ μια κλάση μπορεί να επεκτείνει μόνο μια κλάση βάση, ωστόσο έχει την δυνατότητα να υλοποιεί περισσότερες από μια διαπροσωπείες. Για παράδειγμα θεωρείστε τις διαπροσωπείες Inter1 της παραγράφου 8.5 και την διαπροσωπεία:

```
interface Inter2{
void method4();}
```

οπότε είναι δυνατό να οριστεί η κλάση Example2 που να υλοποιεί και τις δύο αυτές διαπροσωπείες:

```
class Example2: Inter1, Inter2{
public int method1(int m){
if(m<=0)m=m*m;
return m;}
public void method2(){
System.Console.WriteLine("I love C#…");}
public void method4(){
System.Console.WriteLine("I love C# and Java…");}}
```

Η κλάση Example2 αναπτύσσει όλες τις μεθόδους των δύο διαπροσωπειών Inter1, Inter2. Οπωσδήποτε, δεν συμβουλεύουμε τον αναγνώστη, στη συχνή χρήση τέτοιων σχηματισμών κληρονομικότητας. Σε περίπτωση που χρησιμοποιούνται τέτοια σχήματα, θα πρέπει να δοθεί προσοχή στα οριζόμενα αντικείμενα μέσα στα προγράμματα. Αν θα ορίζονται στο ένα Interface ή στο άλλο, ή και σε κλάση, που υλοποιεί άμεσα κάποιο από αυτά ή και τα δύο. Για παράδειγμα αν δοθεί:

```
Inter1 c= new Example1();
```

Τότε το αντικείμενο c μπορεί να αντιπροσωπεύει τις μεθόδους της διαπροσωπείας Inter1. Ωστόσο, μπορεί να δοθεί στη συνέχεια:

```
Example2 d= new Example2();
c=d;
```

οπότε το αντικείμενο c έχει τη δυνατότητα αναφοράς σε όλες τις μεθόδους των διαπροσωπειών Inter1 και Inter2.

8.9 ΠΑΡΕΧΟΜΕΝΑ ΠΑΚΕΤΑ ΣΤΗ C#

Είναι γνωστό στο χρήστη της Java, ότι συνοδεύεται από δεκάδες παρεχόμενων κλάσεων και διαπροσωπειών (interfaces) όπου το καθένα από αυτά έχει πολλές φορές, και εκατοντάδες παρεχόμενες μεθόδους. Αυτό δεν σημαίνει, βέβαια, ότι καλύπτει τις συχνές απαιτήσεις των χρηστών, μάλιστα

υπάρχουν περιπτώσεις που δεν καλύπτονται ούτε οι πλέον συνηθισμένες προγραμματιστικές απαιτήσεις. Ιδιαίτερα στα θέματα προσπέλασης δεδομένων από αρχεία, αλλά και από κονσόλα, η Java διαθέτει ένα ισχυρό πλήθος από steams δεδομένων με δεκάδες μεθόδους, χωρίς όμως να καλύπτει αντίστοιχα ορισμένες βασικές ανάγκες.

Ευτυχώς, η C# έχει λύσει ορισμένα από τα προβλήματα αυτά παραπέμποντας τα περισσότερα στη βασική κλάση **System.** Στη συνέχεια θα δοθούν ιδιαίτεροι πίνακες με τις βασικές κλάσεις της Java σε αντιστοιχία με εκείνες της C#.

8.9.1 ΤΟ ΠΑΚΕΤΟ java.lang

Πίνακας 8.9.1 : Σύγκριση του Java πακέτου java.lang με τη C#

Java	C#
Boolean	System.Boolean
Byte	System.Byte
Character	System.Char
Class	System.Type
ClassLoader	N/A
Compiler	N/A
Double	System.Double
Float	System.Single
InheritableThreadLocal	N/A
Integer	System.Int32
Long	System.Int64
Math	System.Math
Number	N/A
Object	System.Object
Package	N/A
Process	System.Diagnostics.Process
Runtime	System.Diagnostics.Process
SecurithManager	N/A
Short	System.Int16
StackTraceElement	N/A
StrictMath	System.Math
String	System.String
StringBuffer	System.Text.StringBuilder
System	N/A
ThreadGroup	N/A
ThreadLocal	System.LocalDataStoreSlot
Throwable	System.Exception
Void	System.Void
PhantomReference	N/A
Reference	N/A
ReferenceQueue	N/A
SoftReference	N/A
WeakReference	System.WeakReference
AccesibleObject	System.Reflection.MemberInfo
Array	System.Array
Constructor	System.Reflection.ConstructorInfo
Field	System.Reflection.FieldInfo

Method	System.Reflection.MethodInfo
Modifier	System.Reflection.MethodAttributes
Proxy	N/A
ReflectPermission	System.Security.Permissions.ReflectionPermission

8.9.2 ΤΟ ΠΑΚΕΤΟ java.math

Java	C#
BigDecimal	System.Decimal
BigInteger	N/A

8.9.3 ΤΟ ΠΑΚΕΤΟ java.net

Java	C#
Authenticator	System.Net.AuthenticationManager
ConentHandler	N/A
DatagramPacket	N/A
DatagramSocket	System.Net.Sockets.UdpClient
DatagramSocketImpl	N/A
HttpURLConnection	System.Net.WebClient System.Web.WebRequest System.Web.WebResponse
Inet4Address	System.Net.IPAddress
Inet6Address	System.Net.IPAddress
InetAddress	System.Net.IPAddress
InetSocketAddress	System.Net.IPEndPoint
JarURLConnection	N/A
MulticastSocket	System.Net.Sockets.UdpClient
NetPermission	System.Ne.SocketPermission System.Net.WebPermission
NetworkInterface	N/A
PasswordAuthentication	System.Net.NetworkCredential
ServerSocket	System.Net.Sockets.TcpListener
Socket	System.Net.Sockets.TcpClient
SocketAddress	System.Net.SocketAddress
SocketImpl	N/A
SocketPermission	System.Net.SocketPermission System.Net.WebPermission
URI	System.Uri
URL	System.Uri
URLClassLoader	N/A
URLConnection	System.Net.WebClient System.Web.WebRequest System.Web.WebResponse
URLDecoder	System.Uri
URLEncoder	System.Uri
URLStreamHadler	N/A

8.9.4 ΤΟ ΠΑΚΕΤΟ java.sql

Java	C#
Διασύνδεση (interface)	
Array	N/A
Blob	Το Blob διαβάζεται από το DataReader,δείτε τύπους: System.Data.IdbDataReader System.Data.SqlClient.SqlDataReader System.Data.OleDb.OleDbDataReader
CallableStatement	System.Data.IdbCommand System.Data.SqlClient.SqlCommand System.Data.OleDb.OleDbCommand
Clob	Το αντικείμενο Clob διαβάζεται μέσα από το DataReader, δείτε τους παρακάτω τύπους : System.Data.IdbDataReader System.Data.SqlClient.SqlDataReader System.Data.OleDb.OleDbDataReader
Connection	System.Data.Idb System.Data.SqlClient.Sql System.Data.OleDb.OleDb
DatabaseMetaData	N/A
Driver	N/A
ParameterMetaData	System.Data.IDataParameter System.Data.IDbDataParameter System.Data.IDataParameterCollection System.Data.SqlClient.SqlParameter System.Data.SqlClient.SqlParameterCollection System.Data.OleDb.OleDbParameter System.Data.OleDb.OleDbParameterCollection
PreparedStatement	System.Data.IDbCommand System.Data.SqlClient.SqlCommand System.Data.OleDb.OleDbCommand
Ref	N/A
ResultSet	System.Data.IdbDataReader System.Data.SqlClient.SqlDataReader System.Data.OleDb.OleDbDataReader
ResultSetMetaData	System.Data.IdbDataReader System.Data.SqlClient.SqlDataReader System.Data.OleDb.OleDbDataReader System.Data.DataSet
Savepoint	System.Data.SqlClient.SqlTransaction
SQLData	N/A
SQLInput	N/A
SQLOutput	N/A
Statement	System.Data.IDbCommand System.Data.SqlClient.SqlCommand System.Data.OleDb.OleDbCommand
Struct	N/A
Κλάσεις (Classes)	

Date	System.Data.SqlTypes.SqlDateTime
DriverManager	N/A
DriverPropertyInfo	N/A
SQLPermission	System.Data.Common.DBDataPermission System.Data.Common.DBDataPermissionAttribute System.Data.SqlClient.SqlClientPermission System.Data.SqlClient.SqlClientPermissionAttribute System.Data.OleDb.OleDbPermission System.Data.OleDb.OleDbPermissionAttribute
Time	System.Data.SqlTypes.SqlDateTime
TimeStamp	N/A
Types	System.Data.DbTypes

8.9.5 ΤΟ ΠΑΚΕΤΟ java.text

Java	C#
ChoiseFormat	N/A
DateFormat	System.DateTime System.Globalization.DateTimeFormatInfo
DateFormatSymbols	System.Globalization.DateTimeFormatInfo
FieldPosition	N/A
MessageFormat	N/A (Η διάταξη είναι χτισμένη μέσα στις κλάσεις οι οποίες λειτουργούν τη διασύνδεση System.IFormattable).
NumberFormat	System.Convert System.Globalization.NumberFormatInfo Η διάταξη είναι χτισμένη μέσα στις κλάσεις οι οποίες λειτουργούν τη διασύνδεση System.IFormattable.
ParsePosition	N/A
RuleBasedCollator	N/A
SimpleDateFormat	System.DateTime System.Globalization.DateTimeFormatInfo
StringCharacterIrerator	System.Ienumerator πετυχαίνεται διαμέσου System.String

8.9.6 ΤΟ ΠΑΚΕΤΟ java.util

Java	C#
AbstractCollection	System.Collections.CollectionsBase
AbstractList	N/A
AbstractMap	N/A
AbstractSequentialList	N/A
AbstractSet	N/A
ArrayList	System.Collections.ArrayList
Arrays	System.Array
BitSet	N/A
Calendar	System.Globaliazation.Calendar
Collections	N/A

Currency	System.Globalization.RegionInfo
Date	System.DateTime
Dictionary	System.Collections.DictionaryBase
EventListenerProxy	N/A
EventObject	System.EventArgs
GregorianCalendar	System.Globalization.GregorianCalendar
HashMap	System.Collections.Hashtable
IdentityHashMap	N/A
LinkedHashMAp	N/A
LinkedHashSet	N/A
LinkedList	N/A
ListResourceBundle	System.Resources.ResourceManager System.Resources.ResourceSet
Locate	System.Globalization.RegionInfo System.Globalization.CultureInfo
Observable	N/A
Properties	N/A
PropertyPermission	N/A
PropertyResourceBundle	N/A
Random	System.Random
ResourceBundle	System.Resources.ResourceManager System.Resources.ResourceSet
SimpleTimeZone	System.DateTime
Stack	System.Collections.Stack
StringTokenizer	N/A
Timer	System.Threading.Timer System.Timers.Timer
TimerTask	System.Threading.TimerCallback System.Timers.ElapsedEventHandler
TimeZone	System.DateTime
TreeMap	N/A
TreeSet	System.Collections.SortedList
Vector	System.Collections.ArrayList
WeakHashMap	N/A
java.util.jar	N/A
java.util.logging	N/A
java.util.prefs	N/A (εναλλακτικά isolated storage)

8.9.7 ΤΟ ΠΑΚΕΤΟ java.util.regex

Java	C#
Κλάσεις (classes)	
Matcher	System.Text.RegularExpressions.Regex System.Text.RegularExpressions.Match System.Text.RegularExpressions.MatchCollection System.Text.RegularExpressions.Group System.Text.RegularExpressions.GroupCollection
Pattern	System.Text.RegularExpressions.Regex System.Text.RegularExpressions.RegexComplicationInfo System.Text.RegularExpressions.RegexOptions
Εξαιρέσεις (Exceptions)	

PatternSyntaxException	System.ArgumentException
java.util.zip	N/A

8.9.8 ΤΟ ΠΑΚΕΤΟ javax.swing

Java	C#
AbstractAction	N/A
AbstractButton	System.Windows.Forms.ButtonBase System.Windows.Forms.Button
AbstractCellEditor	N/A
AbstractListModel	System.Windows.Forms.ListControl
AbstractSpinnerModel	System.Windows.FormsUpDownBase
ActionMap	N/A
BorderFactory	N/A
Box	N/A
BoxLayout	N/A
ButtonGroup	N/A
CellRenderPane	N/A
ComponentInputMap	N/A
DebugGraphics	N/A
DefaultBoundedRangeModel	N/A
DefaultButtonModel	N/A
DefaultCellEditor	N/A
DefaultComboBoxModel	N/A
DefaultDesktopManager	N/A
DefaultFocusManager	N/A
DefaultListCellRenderer	N/A
DefaultListModel	N/A
DefaultListSelectionModel	N/A
DefaultSingleSelectionModel	N/A
FocusManager	N/A
GrayFilter	N/A
ImageIcon	System.Drawing.Image
InputMap	N/A
InputVerifier	N/A
JApplet	N/A
JButton	System.Windows.Forms.Button
JCheckBox	System.Windows.Forms.CheckBox
JCheckBoxMenuItem	N/A
JColorChooser	System.Windows.Forms.ColorDialog
JComboBox	System.Windows.Forms.ComboBox
JComponent	System.Windows.Forms.UserControl System.Windows.Forms.Control
JDesktopPane	N/A
JDialog	System.Windows.Forms.CommonDialog
JEditorPane	System.Windows.Forms.TextBoxBase
JFileChooser	System.Windows.Forms.OpenFileDialog System.Windows.Forms.SaveFileDialog
JFormattedTextField	System.Windows.Forms.RichTextBox
JFrame	System.Windows.Forms.Form
JInternalFrame	N/A

JLabel	System.Windows.Forms.Label
	System.Windows.Forms.Link-Label
JLayeredPane	N/A
JList	System.Windows.Forms.ListBox
Jmenu	N/A
JMenuBar	System.Windows.Forms.MainMenu
JMenuItem	System.Windows.Forms.MenuItem
JOptionPane	N/A
Jpanel	System.Windows.Forms.Panel
JPasswordField	System.Windows.Forms.TextBox
JPopupMenu	System.Windows.Forms.ContextMenu
JpopupMenu.Separator	N/A
JProgressBar	System.Windows.Forms.StatusBar
JRadioButton	System.Windows.Forms.RadioButton
JRadioButtonMenuItem	N/A
JRootPane	N/A
JScrollBar	System.Windows.Forms.HScrollBar
	System.Windows.Forms.VScrollBar
JScrollPane	System.Windows.Forms.Panel
JSeparator	N/A
Jslider	System.Windows.Forms.TrackBar
JSpinner	System.Windows.Forms.DomainUpDown
	System.Windows.Forms.NumericUpDown
JSplitPane	System.Windows.Forms.Splitter
JTabbedPane	System.Windows.Forms.TabControl
Jtable	System.Windows.Forms.ListView
JTextArea	System.Windows.Forms.TextBox
JTextField	System.Windows.Forms.TextBox
JTextPane	System.Windows.Forms.RichTextBox
JToggleButton	System.Windows.Forms.ButtonBase
JToolBar	System.Windows.Forms.ToolBar
JToolTip	System.Windows.Forms.ToolTip
Jtree	System.Windows.Forms.ListView
JViewport	N/A
JWindow	N/A
KeyStroke	N/A
LayoutFocusTraversalPolicy	N/A
LookAndFeel	N/A
MenuSelectionManager	N/A
Overlayout	N/A
Popup	System.Windows.Forms.ContextMenu
PopupFactory	N/A
ProgressMonitor	N/A
ProgressMonitorInputStream	N/A
RepaintManager	N/A
ScrollPaneLayout	N/A
SizeRequirements	N/A
SizeSequence	N/A
SortingFocusTraversalPolicy	N/A
SpinnerDateModel	N/A
SpinnerListModel	N/A
SpinnerNumberModel	N/A
Spring	N/A

SpringLayout	N/A
SwingUtilities	N/A
Timer	System.Windows.Forms.Timer
ToolTipManager	N/A
TransferHandler	N/A
UIDefaults	N/A
UIManager	N/A
ViewportLayout	N/A
Javax.swing.border	N/A
Javax.swing.colorchooser	N/A
AncestorEvent	N/A
CaretEvent	System.EventArgs
ChangeEvent	N/A
EventListenerList	N/A
HyperlinkEvent	N/A
InternalFrameAdapter	N/A
InternalFrameEvent	N/A
ListDataEvent	System.EventArgs
ListSelectionEvent	System.EventArgs
MenuDragMouseEvent	N/A
MenuEvent	System.EventArgs
MenuKeyEvent	N/A
MouseInputAdapter	N/A
PopupMenuAdapter	System.EventArgs
SwingPropertyChangeSupport	N/A
TableColumnModelEvent	N/A
TableModelEvent	N/A
TreeExpansionEvent	N/A
TreeModelEvent	N/A
TreeSelectionEvent	N/A
UndoableEditEvent	N/A
Javax.swing.filechooser	N/A εναλλακτικά System.Windows.Forms.OpenFileDialog και System.Windows.Forms.SaveFileDialog
Javax.swing.plaf	N/A
Javax.swing.table	N/A εναλλακτικά System.Windows.Forms.ListBox
Javax.swing.text	N/A εναλλακτικά System.Windows.Forms.RichText
Javax.swing.tree	N/A εναλλακτικά System.Windows.Forms.ListBox
Javax.swing.undo	N/A

8.9.9 ΤΟ ΠΑΚΕΤΟ java.awt

Java	C#
AlphaComposite	N/A
AWTevent	System.EventArgs
AWTEventMultiCaster	N/A
AWTKeyStroke	System.Windows.Forms.KeyPressEventArgs
AWTPermission	System.Security.Permissions.UIPermission
BasicStroke	System.Drawing.Pen

BorderLayout	N/A
BufferCapabilities	N/A
Button	System.Windows.Forms.Button
Canvas	System.Windows.Forms.Control
CardLayout	N/A
Checkbox	System.Windows.Forms.RadioButton
CheckboxGroup	N/A (Όλα τα RadioButton αντικείμενα μέσα στο .NET είναι αμοιβαία εκτελέσιμα μέσα σε έλεγχο.)
CheckboxMenuItem	N/A
Choice	System.Windows.Forms.ComboBox
Color	System.Drawing.Color
Component	System.Windows.Forms.Control
ComponentOrientation	N/A
Container	System.Windows.Forms.Control
Cursor	System.Windows.Forms.Cursor
Dialog	System.Windows.Forms.CommonDialog
Dimension	System.Drawing.Size
DisplayMode	N/A
Event	System.EventArgs
FileDialog	System.Windows.Forms.FileDialog System.Windows.Forms.OpenFileDialog System.Windows.Forms.SavaFileDialog
FlowLayout	N/A
FocusTraversalPolicy	N/A
Font	System.Drawing.Font
FontMetrics	N/A
Frame	System.Windows.Forms.Form
GradientPaint	System.Drawing.Drawing2D.LinearGradientBrush
Graphics	System.Drawing.Graphics
Graphics2D	System.Drawing.Graphics
GraphicsConfigTemplate	N/A
GraphicsConfiguration	N/A
GraphicsDevice	N/A
GraphicsEnviroment	N/A
GridBagConstraints	N/A
GridBagLayout	N/A
GridLayout	N/A
Image	System.Drawing.Image
ImageCapabilities	N/A
Insets	N/A
JobAttributes	System.Drawing.Printing.PageSettings System.Drawing.Printing.PrinterSettings
KeyboardFocusManager	N/A
Label	System.Windows.Forms.Label
List	System.Windows.Forms.ListBox
MediaTracker	N/A
Menu	SystemWindows.Forms.MainMenu
MenuBar	N/A
MenuShortcut	N/A
PageAttributes	System.Drawing.Printing.PageSettings
Panel	System.Windows.Forms.Panel
Point	System.Drawing.Point
Polygon	N/A

PopupMenu	System.Windows.Forms.ContextMenu
PrintJob	System.Drawing.Printing.PrintDocument
Rectangle	System.Drawing.Rectangle
RenderingHints	N/A
RenderingHints.Key	N/A
Robot	N/A
Scrollbar	System.Windows.Forms.HscrollBar System.Windows.Forms.VscrollBar
ScrollPane	System.Windows.Forms.Pane
ScrollPaneAdjustable	N/A
SystemColor	N/A
TextArea	SystemWindows.Forms.TextBox System.Windows.Forms.RichTextBox
TextComponent	N/A
TextField	System.Windows.Forms.TextBox
TexturePaint	System.Drawing.TextureBrush
Toolkit	N/A
Window	N/A
Java.awt.color	N/A
Java.awt.datatransfer	N/A
Java.dnd	N/A

8.9.10 ΤΟ ΠΑΚΕΤΟ java.awt.event

Java	**C#**
ActionEvent	System.EventArgs
AdjustmentEvent	System.Windows.Forms.ScrollEventArgs
AWTEventListenerProxy	N/A
ComponentAdapter	N/A
ComponentEvent	System.EventArgs
ContainerAdapter	N/A
ContainerEvent	System.Windows.Forms.ControlEventArgs
FocusAdapter	N/A
FocusEvent	System.EventArgs
HierarchyBoundsAdapter	N/A
HierarchyEvent	N/A
InputEvent	N/A
InputMethodEvent	N/A
InvocationEvent	N/A
ItemEvent	System.EventArgs
KeyAdapter	N/A
KeyEvent	System.Windows.Forms.KeyPressEventArgs System.Windows.Forms.KeyEventArgs
MouseAdapter	N/A
MouseEvent	System.Windows.Forms.MouseEventArgs
MouseMotionAdapter	N/A
MouseWheelEvent	System.Windows.Forms.MouseEventArgs
PaintEvent	System.Windows.Forms.PaintEventArgs
TextEvent	System.EventArgs
WindowAdapter	N/A
WindowEvent	System.EventArgs

java.awt.font	N/A
java.awt.geom	N/A
java.awt.im	N/A
java.awt.im.spi	N/A
java.awt.image	N/A
java.awt.image.renderable	N/A

8.9.11 TO ΠΑΚΕΤΟ java.awt.print

Java	C#
Book	N/A
PageFormat	System.Drawing.Printing.PageSettings
Paper	System.Drawing.Printing.PaperSize
PrinterJob	System.Drawing.Printing.PrintDocument

8.9.12 TO ΠΑΚΕΤΟ java.io

Java	C#
BufferedInputStream	System.IO.BufferedStream
BufferedOutputStream	System.IO.BufferedStream
BufferedReader	System.IO.StreamReader
BufferedWriter	System.IO.StreamWriter
ByteArrayInputStream	System.IO.MemoryStream
ByteArrayOutputStream	System.IO.MemoryStream
CharArrayReader	System.IO.StreamReader
CharArrayWriter	System.IO.StreamWriter
DataInputStream	System.IO.BinaryReader
DataOutputStream	System.IO.BinaryWriter
File	System.IO.File
FileinputStream	System.IO.FileStream
FileOutPutStream	System.IO.FileStream
FilePermission	N/A
FileReader	System.IO.StreamReader
FileWriter	System.IO.StreamWriter
FilterInputStream	N/A
FilterOutputStream	N/A
FilterReader	N/A
FilterWriter	N/A
InputStream	System.IO.Stream
InputStreamReader	N/A
LineNumberInputStream	N/A
LineNumberReader	N/A
ObjectInputStream	N/A
ObjectOutputStream	N/A
OutputStream	System.IO.Stream
OutputStreamWriter	N/A
PipedInputStream	N/A

PipedOutputStream	N/A
PipedReader	N/A
PipedWriter	N/A
PrintStream	System.IO.StreamWriter
PrintWriter	System.IO.StreamWriter
PushBackInputStream	System.IO.StreamReader
PushBackReader	System.IO.StreamReader
RandomAccessFile	System.IO.FileStream
Reader	N/A
SequenceInputStream	N/A
StreamTokenizer	N/A
StringBufferInputStream	System.IO.StringReader
StringReader	System.IO.StringReader
StringWriter	System.IO.StringWriter
Writer	N/A

8.9.13 ΛΟΙΠΕΣ ΠΑΡΕΧΟΜΕΝΕΣ ΚΛΑΣΕΙΣ

Java	**C#**
jana.nio	N/A
java.rmi	N/A
java.security	System.Security, System.Cryptography
Javax.sql	System.Data

8.10 ΑΣΚΗΣΕΙΣ

1. Έστω το ακόλουθο πρόγραμμα:

```
public class Test
 {
   public static void main(String[] args)
    {
outer:
 for (int i=0;i<3;i++)
  { for (int j=0;j<3;j++)
    { if (i+j==3)
      { System.out.println("i-j="+ Math.abs(i-j));
        continue outer;
      }
      System.out.println("i="+ i+";j="+j);
    }
  }
 }
}
```

Να μεταφρασθεί ο κώδικας σε C# και να προστεθούν τα κατάλληλα πακέτα

2. Έστω ο παρακάτω κώδικας

```
public class Node extends UnicastRemoteObject
implements Rnode
{ public Node() throws RemoteException{}
  public Rnode getNext() throws RemoteException
   { return next;
   }
   public void setNext(Rnode next) throws RemoteException
    { this.next=next;
    }
   public int getData() throws RemoteException
    { return data;
    }
  public void setData(int data) throws RemoteException
   { this.data=data;
   }
  public String getLocation() throws RemoteException
   { return location;
   }
   public void setLocation(String locaton) throws
    RemoteException
    { this.location=location;
    }
    private Rnode next;
    private int data;
    private String location;
 }
```

Να μεταφρασθεί ο κώδικας σε C# και να προστεθούν τα κατάλληλα πακέτα

ΚΕΦΑΛΑΙΟ 9ο

9. ΑΡΧΕΙΑ

9.1 ΕΙΣΑΓΩΓΗ

Η **είσοδος** και η **έξοδος (Input, Output, I/O)** γίνεται στις διαδικτυακές γλώσσες προγραμματισμού με ειδικές παρεχόμενες μεθόδους μέσα από τις κλάσεις των γλωσσών αυτών.

Στη Java οι βασικές κλάσεις, προσπέλασης δεδομένων από αρχεία ή από δίκτυο, ή από οποιοδήποτε άλλο φορέα γίνονται από οκτώ βασικές υποκλάσεις τις: **object,** καθώς επίσης με παράλληλη ανάπτυξη ειδικών παρεχόμενων μεθόδων, διαπροσωπειών (interfaces). Οι σημαντικότερες από αυτές είναι οι **InputStream** (βασικά για αναζήτηση binary δεδομένων), ή **OutputStream** (αντίστοιχη της InputStream για εγγραφή δεδομένων) με αντίστοιχες υποκλάσεις την **Reader,** και την **Writer** για περιπτώσεις ανάγνωσης/ εγγραφής δεδομένων σε χαρακτήρες. Ακόμη, πρέπει επίσης να αναφερθούν οι υποκλάσεις **File** για διαχείριση αρχείων καθώς και η **RandomAccessFile** για τυχαία προσπέλαση του **binary αρχείου.** Σε κάθε περίπτωση θα έπρεπε να δημιουργηθεί ένα ειδικό αντικείμενο προσπέλασης το επονομαζόμενο **IOStreams**. Τα ρεύματα αυτά χωρίζονται σε δύο μεγάλες κατηγορίες:

- **BS (ByteStreams)** ρεύματα δεδομένων που οι αριθμητικές τιμές δεν είναι εκφρασμένες σε χαρακτήρες, αλλά είναι όπως είναι γνωστές στον επεξεργαστή σε μορφή **bits.** Αντιστοιχούν στα **binary αρχεία**
- **CS (CharacterStreams)** ρεύματα δεδομένων σε **χαρακτήρες** αντιστοιχούν σε **αρχεία κειμένου (text files)**

Η C# συνεχίζει να διακρίνει και αυτή τα ρεύματα σε **BS, CS**. Ωστόσο, διαθέτει δικές της κλάσεις για προσπέλαση δεδομένων. Τις περισσότερες μεθόδους, τις παίρνει μέσα από την κλάση **SystemIO** ενώ σχεδόν όλα τα υπόλοιπα όπως είναι η διαχείριση αρχείων, μέσα από τη κλάση **FileStream**.

9.2 ΧΑΡΑΚΤΗΡΙΣΤΙΚΕΣ ΠΡΟΣΠΕΛΑΣΕΙΣ

Τους μηχανικούς λογισμικού, δεν τους ενδιαφέρουν όμοια όλα τα είδη προσπελάσεων. Στην πράξη, φαίνεται πως οι I/O λειτουργίες που εμφανίζουν το μεγαλύτερο ενδιαφέρον είναι οι ακόλουθες:

- **Επίπεδο οθόνης:** όταν οι λειτουργίες I/O γίνονται στην οθόνη του υπολογιστή μας, η οποία χωρίζεται σε δύο μεγάλες κατηγορίες, σε κονσόλα των 32 bits και σε επίπεδο παραθύρων. Ήδη σε προηγούμενα κεφάλαια μέσα στα πλαίσια παραδειγμάτων έχουν χρησιμοποιηθεί δύο-τρεις τεχνικές για I/O σε κονσόλα, ενώ η τεχνική με παράθυρα θα αναλυθεί σε επόμενο κεφάλαιο. Το επίπεδο οθόνης είναι χρήσιμο όταν ο χρήστης συμμετέχει στα δεδομένα ή αποτελέσματα, στη διάρκεια της εκτέλεσης του προγράμματος, όπως στην **παραμετροποίηση** και **on line data entry**.

- **Σειριακή Προσπέλαση Αρχείων:** η απλούστερη μορφή αρχείων έχει σαν βασικό συστατικό την **γραμμή (line)**, η οποία συγκροτείται από **χαρακτήρες.** Οι γραμμές δίνουν το **κείμενο (text)** και τελικά ένα τέτοιο αρχείο ονομάζεται **αρχείο κειμένου (text file).** Συχνά επίσης, ένα τέτοιο αρχείο αναφέρεται και σαν αρχείο **ASCII, document, Formated File.** Προσοχή, όμως, τα αρχεία κειμένου της C# ακολουθούν τους χαρακτήρες του πρότυπου Unicode, οι οποίοι φτάνουν σε πλήθος τους 70000 περίπου. Παρ' όλες αυτές τις μικρές διαφορές, στα αρχεία κειμένου που μπορεί να έχε η C# από τη Java, σχεδόν πάντα η προσπέλαση σε αρχεία γίνεται σειριακά, δηλαδή η προσπέλαση της m γραμμής προϋποθέτει προσπέλαση όλων των προηγουμένων γραμμών.(**Sequential Access**)
- **Κατευθυνόμενη Προσπέλαση Αρχείων:** όπου το βασικό συστατικό του αρχείου είναι η **εγγραφή (record)** η οποία συγκροτείται από **bits**, από αριθμητικές τιμές όπως ακριβώς τις γνωρίζει ο ίδιος ο επεξεργαστής. Ένα τέτοιο αρχείο ονομάζεται **by bit representation file** ή **binary file** ή και **unformatted file.** Αρχεία όπως *.exe, dbms και κάθε data base file είναι σχεδόν binary αρχεία, τα οποία έχουν το πλεονέκτημα της ελαχιστοποίησης περιφερειακής μνήμης, αλλά και χρόνου προσπέλασης. Επιτρέπουν την **κατευθυνόμενη προσπέλαση (direct access)** σε επίπεδο εγγραφής ή και σε επίπεδο τοποθέτησης (file position). Η C# ακολουθεί την τεχνική του **file position** η οποία είναι αγαπητή περισσότερο σε γλώσσες λογικού προγραμματισμού (πχ Prolog), αν και υπάρχει πάντα μαθηματική σχέση που οδηγεί από τη μια τεχνική στη άλλη. Στην κατευθυνόμενη διαδικασία, όταν ακολουθείται η τεχνική του file position, η όλη διαδικασία ονομάζεται **Random Access**, όπου γίνεται άμεση προσπέλαση στο k-byte του αρχείου και μετά, αρχίζοντας τη μέτρηση από την αρχή ή από το τέλος, ή και από προηγούμενη θέση προσπέλασης. Τέλος, οι μέθοδοι που υποστηρίζουν την τυχαία προσπέλαση ανήκουν στο low level programming και η C# διαθέτει πολλές από αυτές.

Βέβαια, εκτός από τα επίπεδα λειτουργιών I/O που αναφέρθηκαν, υπάρχουν και άλλες που σχετίζονται με τα περιφερειακά (devices) που χρησιμοποιούνται, όπως τα **sockets (αμφίδρομος δίαυλος επικοινωνίας μεταξύ δύο προγραμμάτων που κινούνται στο δίκτυο)** και τις αποθήκες μνήμης **(memory buffers).**

9.3 ΡΕΥΜΑΤΑ ΧΑΡΑΚΤΗΡΩΝ

Στη Java η κονσόλα εκτίθεται στον προγραμματιστή μέσα από ένα σύνολο από streams που βρίσκονται μέσα στην βιβλιοθήκη **java.lang.System.** Από την άλλη μεριά, το .NET Framework τοποθετεί όλες τις συναρτήσεις της κονσόλας στη βιβλιοθήκη **System.Console.** Η κλάση Console είναι ένα μείγμα από τρία streams τα οποία αποτελούνται από read only properties. Οι μέθοδοι αυτοί φαίνονται στον παρακάτω πίνακα:

JAVA	**C#**
System.out	Console.Out
System.in	Console.In
System.err	Console.Error

Τα error και out properties επιστρέφουν τιμές μέσω του **System.IO.TextWriter** ενώ αντίθετα το in property επιστρέφει ένα στιγμιότυπο του **System.IO.TextReader.**

9.3.1 ΕΓΓΡΑΦΗ-ΑΝΑΓΝΩΣΗ ΑΠΟ ΚΟΝΣΟΛΑ

Η εγγραφή σε κονσόλα είναι παρόμοια σε λειτουργικότητα με το **java.io.PrintStream** για την ακρίβεια η **TextWriter.Write** μέθοδος φορτώνεται και δέχεται απλούς τύπους string οι οποίοι εμφανίζονται στην κονσόλα. Ένα απλό παράδειγμα είναι το παρακάτω.

```
Console.Out.WriteLine(“this is a message”) ;
Console.WriteLine(“this is a message”) ;
Console.Out.Write(“this is a message”) ;
Console.Write(“this is a message”) ;
```

Για να διαβάσουμε από κονσόλα, χρησιμοποιούμε το System.IO.TextReader ένα απλό παράδειγμα για να γίνει κατανοητό είναι το παρακάτω:

```
Console.In.ReadLine();
Console.ReadLine();
Console.In.Read();
Console.Read();
```

Το ReadLine διαβάζει την επόμενη γραμμή χαρακτήρων από το input stream ψάχνοντας να βρει τον διαχωριστή γραμμής. Ενώ το read επιστρέφει ένα απλό χαρακτήρα και αυτό μόνο όταν έχει τελειώσει την ανάγνωση και έχει πατηθεί το enter key (πλήκτρο enter).

Συνοπτικά, η κλάση Console αντιστοιχίζεται στην Java όπως φαίνεται στον παρακάτω πίνακα.

JAVA	**C#**
System.in	Console.In
System.out	Console.Out
System.err	Console.Error
System.out.println()	Console.WriteLine() Console.Out.WriteLine()
System.out.print()	Console.Write() Console.Out.Write()
System.in.read()	Console.Read() Console.In.Read()
N/A	Console.ReadLine() Console.In.ReadLine()
System.err.print()	Console.Error.Write()
System.err.println()	Console.Error.WriteLine()
System.setIn()	Console.SetIn()
System.setOut()	Console.SetOut()
System.setErr()	Console.SetError()

Ένα πρόγραμμα εισόδου από κονσόλα είναι το παρακάτω γραμμένο σε Java:

```
import java.io.*;

public class consola
 { //reading a double value
   public double c_double() throws IOException
    { return (Double.parseDouble(c_string()));
    }
   //reading a string
   public String c_string() throws IOException
    { BufferedReader buffer= new BufferedReader(new
      InputStreamReader(System.in));
      String str=null;
      str=buffer.readLine();
      return str;
    }
    // reading a long value
    public long c_long() throws IOException
     { return((long)(c_double()));
     }
    //reading an Int value
    public int c_int() throws IOException
     { return((int)(c_double()));
     }
    //reading a float value
    public float c_float() throws IOException
     { return((float)(c_double()));
     }
   //reading a short value
   public short c_short() throws IOException
    { return((short)(c_double()));
    }
   //just for testing
   public static void main(String x[]) throws IOException
    { consola obj=new consola();
      System.out.println("give a string:");
      System.out.println("the string is:"+ obj.c_string());
      System.out.println("give a double:");
      System.out.println("the double is:"+ obj.c_double());
      System.out.println("give a float:");
      System.out.println("the float is:"+ obj.c_float());
      System.out.println("give a long:");
      System.out.println("the long is:"+ obj.c_long());
      System.out.println("give an integer:");
      System.out.println("the integer is:"+ obj.c_int());
      System.out.println("give a short:");
      System.out.println("the short is:"+ obj.c_short());
    }
 }
```

Και το αντίστοιχο σε C#.

```
using System;
using System.IO;
public class consola
  {
    public virtual double c_double()
      { throw new System.IO.IOException();
        return (System.Double.Parse(c_string()));
      }
    public virtual System.String c_string()
     { throw new System.IO.IOException();
       System.IO.StreamReader buffer = new
 System.IO.StreamReader((System.Console.In.Read().ToString()));
       System.String str = null;
       str = buffer.ReadLine();
       return str;
       }
    public virtual long c_long()
      { throw new System.IO.IOException();
        return ((long) (c_double()));
      }
     public virtual int c_int()
      {  throw new System.IO.IOException();
         return ((int) (c_double()));
       }
    public virtual float c_float()
      {  throw new System.IO.IOException();
         return ((float) (c_double()));
      }
    public virtual short c_short()
      { throw new System.IO.IOException();
        return ((short) (c_double()));
      }
    [STAThread]
    public static void  Main(System.String[] x)
      {  consola obj = new consola();
         Console.WriteLine("give a string:");
         Console.WriteLine("the string is:"+ obj.c_string());
         Console.WriteLine("give a double:");
         Console.WriteLine("the double is:"+ obj.c_double());
         Console.WriteLine("give a float:");
         Console.WriteLine("the float is:"+ obj.c_float());
         Console.WriteLine("give a long:");
         Console.WriteLine("the long is:"+ obj.c_long());
         Console.WriteLine("give an integer:");
         Console.WriteLine("the integer is:"+ obj.c_int());
```

```
            Console.WriteLine("give a short:");
            Console.WriteLine("the short is:"+ obj.c_short());
     }
    }
```

9.4 ΡΕΥΜΑΤΑ ΧΑΡΑΚΤΗΡΩΝ

Στη Java η κλάση που μας ενδιαφέρει για να διαβάσουμε χαρακτήρες από την οθόνη και όχι, βέβαια ByteStream είναι **BufferedReader** όπου αρχικοποιείται **BufferdReader(Reader object_Reader)**.

Στη Java και στη C# χρησιμοποιούνται κατά κανόνα δύο τύποι ρευμάτων. Ο πρώτος ονομάζεται **character stream (ρεύματα χαρακτήρων)** και χρησιμοποιείται για προσπελάσεις I/O με κύριο στοιχείο τους χαρακτήρες με βασικό συστατικό τη γραμμή, που όταν σχετίζονται με δίσκους, αναφέρονται σε αρχεία κειμένου. Χρησιμοποιούν χαρακτήρες από το πρότυπο **Unicode** και έχουν ήδη διεθνοποιηθεί (internationalized). Τα ρεύματα χαρακτήρων χρησιμοποιούνται μόνο σε high level programming και δεν έχουν ούτε μια κλάση για προγραμματισμό χαμηλού επιπέδου (low level programming). Τέλος, είναι φανερό ότι, τα ρεύματα χαρακτήρων είναι ιδανικά για σειριακές προσπελάσεις αρχείων κειμένου.

Java	**C#**
BufferedReader	System.IO.StreamReader
BufferedWriter	System.IO.StreamWriter
CharArrayReader	System.IO.StreamReader
CharArrayWriter	System.IO.StreamWriter
FileReader	System.IO.StreamReader
FileWriter	System.IO.StreamWriter
FilterReader	N/A
FilterWriter	N/A
InputStreamReader	N/A
LineNumberReader	N/A
OutputStreamWriter	N/A
PipedReader	N/A
PipedWriter	N/A
PrintWriter	System.IO.StreamWriter
PushBackReader	System.IO.StreamReader
Reader	N/A
StringReader	System.IO.StringReader
StringWriter	System.IO.StringWriter
Writer	N/A

Όπως φαίνεται στον πίνακα, υπάρχουν αρκετές παρεχόμενες κλάσεις για ρεύματα χαρακτήρων για τη Java, οι οποίες είναι υποκλάσεις των δύο αφηρημένων κλάσεων **Reader** και **Writer.** Αντίθετα στη C# υπάρχουν μόνο δύο βασικές κλάσεις για ρεύματα χαρακτήρων οι **System.IO.StreamReader** και **System.IO.StreamWriter.**

9.5 ΡΕΥΜΑΤΑ ΜΕ BYTES

Στον αντικειμενοστρεφή προγραμματισμό έχουν επικρατήσει τα **ψηφιακά δεδομένα (binary data)** να ονομάζονται **δεδομένα με bytes.** Στην ουσία πρόκειται για ακέραιες τιμές από 0- 255, όπου όλα τα είδη δεδομένων μπορούν τελικά να εκφραστούν με μια τέτοια μορφή, άμεσα ή με συνδυαστικές μεταξύ τους. Μια τέτοια ροή I/O ονομάζεται στην Java αλλά και στη C# **ρεύμα με bytes (byte stream(BS))** και τονίζεται πως πρόκειται για το σύνολο των λογισμικών προϊόντων (RDBMS, εκτελέσιμα προγράμματα κλπ) πλην των αρχείων χαρακτήρων

Στα ρεύματα με bytes, οι κλάσεις **InputStream, OutputStream,** είναι οι δύο αφηρημένες κλάσεις που διαθέτουν μεθόδους για ανάγνωση και εγγραφή μιας ροής δεδομένων με bytes στη Java. Στη C# χρησιμοποιείται το πακέτο **System.IO** ενώ για **κατευθυνόμενη διαδικασία** χρησιμοποιείται η μέθοδος **Seek** του πακέτου **System.IO.FileStream.**

JAVA	**C#**
BufferedInputStream	System.IO.BufferedStream
BufferedOutputStream	System.IO.BufferedStream
ByteArrayInputStream	System.IO.MemoryStream
ByteArrayOutputStream	System.IO.MemoryStream
DataInputStream	System.IO.BinaryReader
DataOutputStream	System.IO.BinaryWriter
File	System.IO.File
FileinputStream	System.IO.FileStream
FileOutPutStream	System.IO.FileStream
FilePermission	N/A
FilterInputStream	N/A
FilterOutputStream	N/A
InputStream	System.IO.Stream
LineNumberInputStream	N/A
ObjectInputStream	N/A
ObjectOutputStream	N/A
OutputStream	System.IO.Stream
PipedInputStream	N/A
PipedOutputStream	N/A
PrintStream	System.IO.StreamWriter
PushBackInputStream	System.IO.StreamReader
RandomAccessFile	System.IO.FileStream
SequenceInputStream	N/A
StreamTokenizer	N/A
StringBufferInputStream	System.IO.StringReader

Τονίζεται ότι, σε low level programming, κάθε τι παρεχόμενο είναι εκφρασμένο και προσαρμοσμένο σε byte streams.

9.6 I/O ΑΠΟ ΚΟΝΣΟΛΑ

Όπως ήδη αναλύθηκε, το αντικείμενο **System.in** στη Java και αντίστοιχα στη C# το **Console.In** αναφέρονται στην παρεχόμενη κλάση **InputStream** και **File.Stream** αντίστοιχα οι οποίες λειτουργούν με byte stream. Επιπρόσθετα, τα ίδια αυτά αντικείμενα χρησιμοποιούνται για στάνταρ είσοδο από κονσόλα. Όμως, για να γίνει ανάγνωση δεδομένων από την κονσόλα μας ενδιαφέρει να είναι εκφρασμένη σε Character Stream αντί για BS και επιπρόσθετα να υπάρχει προσωρινή αποθήκευση των χαρακτήρων εισόδου. Άρα η κλάση που μας ενδιαφέρει είναι η **System.IO.StreamReader**, αρκεί να οριστεί ένα αντικείμενο με κατάλληλη αρχικοποίηση. Μια αποτελεσματική κατασκευαστική μέθοδος αυτής της κλάσης έχει το συντακτικό τύπο:

StreamReader (StreamReader object_Reader)

Όπου το **object_Reader** αναφέρεται στην κλάση **Reader** η οποία όμως είναι αφηρημένη και επιπρόσθετα πρέπει τα bytes του **Console.In** να μετατραπούν σε χαρακτήρες. Στο σημείο αυτό χρησιμοποιείται η κλάση **System.Console.In.Read()** για την μετατροπή αυτή. Άρα ισχύει το αντικείμενο:

new StreamReader(System.Console.In.Read().ToString())

το οποίο μετατρέπει το BS σε CS. Αρκεί, λοιπόν, στη συνέχεια να οριστεί αντικείμενο για προσωρινή αποθήκευση των χαρακτήρων οπότε τελικά έχουμε ένα αντικείμενο που αναφέρεται στην κονσόλα με δεδομένα πληκτρολογίου σε χαρακτήρες

```
System.IO.StreamReader buffer = new
    System.IO.StreamReader((System.Console.In.Read().ToString()
    ));
```

Βέβαια, η προσπάθεια ανάγνωσης δεδομένων έχει και συνέχεια, διότι πρέπει να χρησιμοποιηθεί και η κατάλληλη μέθοδος ανάγνωσης αυτών με τη βοήθεια του αντικειμένου **chr.** Αν θέλουμε να κάνουμε ανάγνωση, πολλών χαρακτήρων τότε γίνεται χρήση του παρακάτω κώδικα:

```
string str;
str=chr.ReadLine();
```

Το παρεχόμενο αντικείμενο Console.In υπακούει αυτόματα ανά γραμμή, δηλαδή, δεν αναγνωρίζει κανένα δεδομένο μέχρις ότου ο χρήστης δώσει enter του πληκτρολογίου. Άρα δεν έχουμε ανάγνωση σε επίπεδο χαρακτήρα, οπότε δεν έχει μεγάλη χρησιμότητα η παρεχόμενη μέθοδος Read(). Αντίθετα η μέθοδος ReadLine() μπορεί να χρησιμοποιηθεί και για ανάγνωση όχι μόνο strings, αλλά και άλλων αριθμητικών τιμών από την κονσόλα.

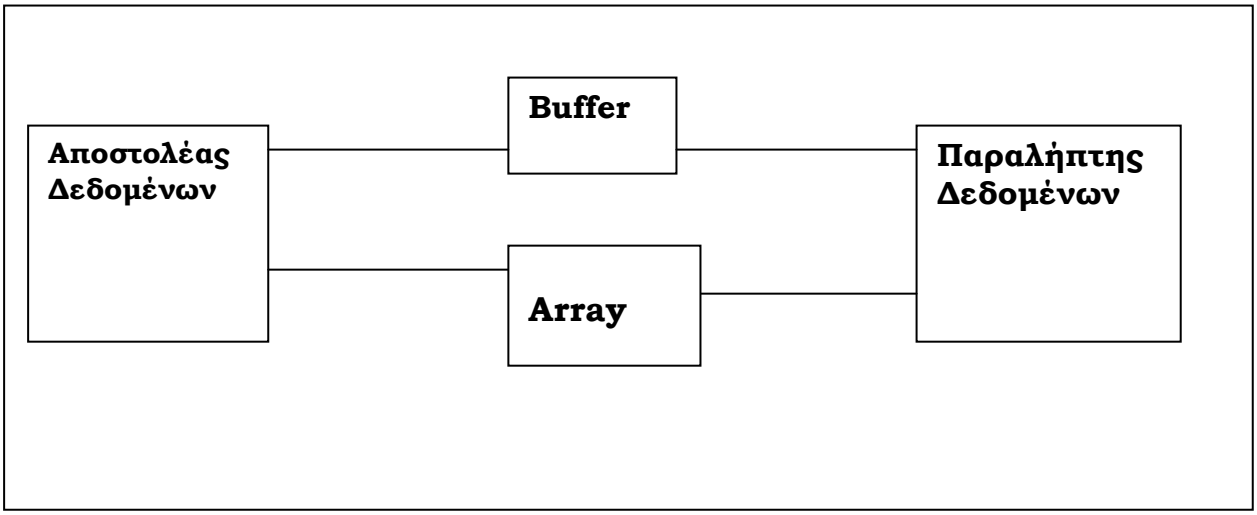
9.7 ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ ΡΕΥΜΑΤΩΝ

Οποιοδήποτε ρεύμα δεδομένων και αν χρησιμοποιεί ο χρήστης, η διαδικασία δημιουργίας και χρήσης βασικά είναι η ίδια. **Για ένα ρεύμα εισόδου, απαιτείται η δημιουργία αντικειμένου που να σχετίζεται με την**

πηγή των data. Στη συνέχεια, μπορεί να γίνει χρήση κάποιας παρεχόμενης μεθόδου για ανάγνωση των δεδομένων μας. Όταν αυτά τελειώσουν, πρέπει το ρεύμα να κλείσει λογικά, συνήθως με κάποια παρεχόμενη μέθοδο του τύπου: **Close().** Αντίστοιχα, για την έξοδο αποτελεσμάτων, θα πρέπει να δημιουργηθεί ένα αντικείμενο που να σχετίζεται με τον προορισμό των δεδομένων. Να γίνει χρήση κάποιας παρεχόμενης μεθόδου για αποστολή (εγγραφή)των δεδομένων μας στην έξοδο και στη συνέχεια να κλείσει λογικά το ρεύμα.

Ένα **φίλτρο** είναι ένας τύπος ρεύματος που τροποποιεί τον αυτόματο χειρισμό του κλασσικού ρεύματος. Η διαδικασία για χρήση φίλτρου είναι απλή και αρχίζει, βέβαια, με τη δημιουργία αντικειμένου που σχετίζεται με την πηγή ή τον προορισμό των δεδομένων, δηλαδή με το βασικό μας ρεύμα. Στη συνέχεια, γίνεται συσχέτιση του φίλτρου με το ρεύμα αυτό. Επομένως, **στο πρόγραμμά μας κάθε ανάγνωση ή εγγραφή μπορεί να γίνει στο φίλτρο αντί στο κλασσικό ρεύμα.** Ο αναγνώστης δεν πρέπει να ανησυχεί διόλου για παρεχόμενες μεθόδους I/O φίλτρων, διότι κατά κανόνα ταυτίζονται με εκείνες των ρευμάτων. Επίσης, είναι δυνατή η χρήση πολλών φίλτρων σε διάφορα επίπεδα διαστρωμάτωσης, ώστε να επιτευχθούν οι κύριοι συνδυασμοί εισόδου ή εξόδου. Για παράδειγμα, το ρεύμα εισόδου να περιέχει αρχείο κειμένου στα ισπανικά, το οποίο διαμέσου φίλτρου να μεταφράζεται στα αγγλικά από το οποίο να απορρίπτονται λέξεις κλειδιά που επιθυμεί ο χρήστης.

Στον προγραμματισμό, η έννοια της **προσωρινής αποθήκευσης** είναι από τις πιο βασικές διότι ο παραλήπτης μπορεί να έχει διαφορετικές ικανότητες επεξεργασίας από τον αποστολέα. **ο ενταμιευτής (buffer)** είναι περιοχή αποθήκευσης όπου κρατούνται δεδομένα σωρευτικά, πριν τα χρειαστεί ένα πρόγραμμα που διαβάζει ή και γράφει δεδομένα. Με τον τρόπο αυτό το πρόγραμμα ξεχνά την αρχική πηγή των δεδομένων ή τον τελικό προορισμό και χρησιμοποιεί άμεσα τον ενταμιευτή. Επίσης, είναι δυνατόν, αντί για απλό ενταμιευτή να χρησιμοποιείται **μήτρα** από bytes ή από χαρακτήρες για αποθήκευση δεδομένων. Σε κάθε περίπτωση η φιλοσοφία είναι η ίδια. Αντί για το αρχικό ρεύμα δεδομένων, χρησιμοποιείται κάτι ενδιάμεσο που το αποθηκεύει. Με τον απλό αυτό τρόπο η C# γλιτώνει χρόνο εσωτερικής συνεννόησης και συντονισμού μεταξύ αποστολέα και παραλήπτη.



Τα ρεύματα εισόδου-εξόδου μας ενδιαφέρουν στην περίπτωση προσπέλασης αρχείων. Ο πίνακας που παρατίθεται δείχνει την αντιστοιχία των δύο γλωσσών.

JAVA	**C#**
FileInputStream FileOutputStream	System.IO.FileStream
ByteArrayInputStream ByteArrayOutputStream	System.IO.MemoryStream
N/A	System.Net.Sockets.NetworkStream

Η βασική κλάση της C# System.IO.FileStream διαθέτει ήδη ενσωματωμένο ενταμιευτή επομένως, δεν χρειάζεται να χρησιμοποιηθεί κάποιο BufferStream, αντίθετα η κλάση System.Nte.Socketss.NetWorkStream δεν διαθέτει εσωτερικό buffer.

9.8 ΔΙΑΧΕΙΡΙΣΗ ΑΡΧΕΙΩΝ-ΚΑΤΑΛΟΓΩΝ

Στη Java, η κλάση file υλοποιεί κάθε διαχείριση σε επίπεδο αρχείου ή καταλόγου. Στη C# το αποτέλεσμα είναι τελικά το ίδιο με τη Java στο θέμα αυτό με δεδομένες κλάσεις του .NET. Η υποκλάση **System.IO.Path** περιλαμβάνει διάφορες μεθόδους σχετικά με τα ονόματα αρχείων σε επίπεδο πληρότητας **(FullPathName)** και καταλόγων. Προσοχή, όμως, δεν κάνει κανένα έλεγχο αν υπάρχει ο υπό όψιν κατάλογος.

Path Class

Member	**Επεξήγηση**
DirectorySeparatorChar	Χαρακτήρας διαχωρισμού καθορισμένος από την πλατφόρμα
InvalidPathChars	Δημιουργία πίνακα απαγορευμένων χαρακτήρων
PathSeparator	Καθορισμός διαχωριστή μονοπατιών
VolumeSeparatorChar	Καθορισμός διαχωριστή συσκευής αποθήκευσης
ChangeExtension()	Αλλαγή επέκτασης
Combine()	Συνένωση δύο μονοπατιών
GetDirectoryName()	Εμφάνιση περιεχομένων μονοπατιού ανάμεσα στο πρώτο και το τελευταίο στιγμιότυπο της DirectorySeparatorChar
GetExtension()	Επιστροφή επέκτασης ενός path
GetFileName()	Επιστροφή περιεχομένων του path μετά το τελευταίο στιγμιότυπο της DirectorySeparatorChar
GetFileNameWithoutExtension()	Επιστροφή των περιεχομένων μονοπατιού μετά το τελευταίο στιγμιότυπο της DirectorySeparatorChar πριν την τελεία
GetFullPath()	Επιστρέφει το πλήρες μονοπάτι για ένα τμήμα μονοπατιού που του δίδεται.

GetPathRoot()	Επιστρέφει τη ρίζα ενός μονοπατιού
GetTempFileName()	Δημιουργεί ένα μηδενικού μήκους προσωρινό αρχείο με μοναδικό path το οποίο και επιστρέφει σαν αποτέλεσμα
GetTempPath()	Επιστρέφει το μονοπάτι του τρέχοντος παροδικού καταλόγου του συστήματος
HasExtension()	Επιστρέφει true αν οι χαρακτήρες του μονοπατιού (path string) περιέχουν το χαρακτήρα περιόδου που ακολουθείται από τουλάχιστον ένα χαρακτήρα
IsPathRooted()	Επιστρέφει true αν το μονοπάτι είναι πλήρως qualified

9.8.1 ΔΙΑΧΕΙΡΙΣΗ ΚΑΤΑΛΟΓΟΥ

Στη C# η κλάση **System.IO.Directory** περιέχει κάθε τι το αναγκαίο για διαχείριση καταλόγου. Είναι ακριβώς το αντίστοιχο με την **System.IO.File** που διαχειρίζεται αρχεία. Η λειτουργία της στηρίζεται στο κάλεσμα μιας μεθόδου της η οποία περιέχει ένα string που αποτελεί το όνομα του επιθυμητού προς επεξεργασία directory. Αντί των δύο παραπάνω μπορούμε να χρησιμοποιήσουμε αντίστοιχα τις **System.IO.DirectoryInfo** και **System.IO.FileInfo,** οι οποίες παρέχουν τις ίδιες λειτουργίες μόνο που απαιτούν να δημιουργήσουν στιγμιότυπο έτσι ώστε να πετύχουμε καλύτερη απόδοση στις λειτουργίες τους. Ένα παράδειγμα για να γίνουν τα παραπάνω κατανοητά είναι το ακόλουθο:

```
FileInfo x_FileInfo= new FileInfo("My_File.txt");
Console.WriteLine(x_FileInfo.CreationTime);
Console.WriteLine(File.GetCreationTime("My_File.txt"));
```

Ένα μικρό μειονέκτημα που έχει η C# το οποίο κληρονόμησε από τη Java είναι η έλλειψη καθορισμού του γεγονότος αν ένα path αντιπροσωπεύει αρχείο ή κατάλογο. Η C# προσπαθεί να καλύψει αυτό το κενό με την μέθοδο **Exists.** Η μέθοδος Exists στην κλάση Directory, θα επιστρέψει true μόνο αν το path αντιπροσωπεύει κατάλογο, ενώ αν το path αντιπροσωπεύει αρχείο, θα επιστρέψει true, μόνο εάν κληθεί μέσα από την κλάση File. Ένα παράδειγμα είναι αυτό που ακολουθεί:

```
string Spath= "C:\\mypath\My_File.txt";
if (Directory.Exists(Spath)){
   Console.WriteLine("{0} is a directory", Spath);}
else if(File.Exists(Spath)){
Console.WriteLine("{0} is a file", Spath);}
else{
Console.WriteLine("{0} Does not exist", Spath);}
```

9.8.2 ΔΙΑΧΕΙΡΙΣΗ ΑΡΧΕΙΩΝ

Η προσπέλαση στα αρχεία προϋποθέτει στη C# την ύπαρξη της βιβλιοθήκης **System.IO.FileStream**. Ο πιο άμεσος τρόπος για να δημιουργήσουμε ρεύματα είναι να δημιουργήσουμε στιγμιότυπο της FileStream στον constructor αυτής. Η βασική μορφή του constructor της FileStream είναι η ακόλουθη:

FileStream(string path, FileMode mode, FileAccess access)

Το FileMode δείχνει πως θα ανοίξει το αρχείο, ειδικότερα:

- **Append**: Άνοιγμα αρχείου, αναζήτηση τέλους ή δημιουργία αν δεν υπάρχει. Χρησιμοποιείται μόνο για εγγραφή.
- **Create:** Δημιουργία φακέλου αν δεν υπάρχει, επικάλυψη του φακέλου αν υπάρχει.
- **CreateNew:** Δημιουργία νέου φακέλου. Αν ο φάκελος υπάρχει εμφανίζεται IOException
- **Open Άνοιγμα αρχείου:** Αν ο φάκελος δεν υπάρχει εμφανίζεται FileNotFoundException.
- **OpenOrCreate:** Αν ο φάκελος υπάρχει εμφανίζεται, αλλιώς τον δημιουργεί.
- **Truncate:** Θέτει το μέγεθος του αρχείου σε μηδέν, και το ανοίγει για γράψιμο

Το FileAccess καθορίζει αν ένα αρχείο θα ανοίξει για **διάβασμα γράψιμο ή και τα δύο** χρησιμοποιώντας την βιβλιοθήκη **System.IO.FileAccess**, η οποία περιέχει τις μεθόδους **Read**, **Write, ReadWrite.** Μερικά παραδείγματα κατανόησης είναι τα ακόλουθα:

```
FileStream("My_File.txt", FileMode.Open, FileAccess.ReadWrite);
FileStream("My_File.txt", FileMode.CreateNew, FileAccess.Read);
FileStream("My_File.txt", FileMode.Truncate, FileAccess.Write);
```

Η πιο πλήρης μορφή του Constructor FileStream είναι η ακόλουθη:

FileStream(string path, FileMode mode, FileAccess access, FileShare share, int buffer, bool async);

- **FileShare:** Καθορίζει πως άλλες διεργασίες έχουν πρόσβαση στο αρχείο το οποίο έχει ανοιχτεί
- **int buffer:** Το μέγεθος του ενταμιευτή
- **bool async:** Όταν έχουμε ασύγχρονο I/O παίρνει την τιμή true

9.8.3 ΔΗΜΙΟΥΡΓΙΑ ΕΜΜΕΣΟΥ ΡΕΥΜΑΤΟΣ

Στιγμιότυπο της FileStream μπορούμε να δημιουργήσουμε έμμεσα μέσω των κλάσεων File και FileInfo. Ο παρακάτω πίνακας παρουσιάζει αυτές τις έμμεσες μεθόδους.

Method	**Περιγραφή**
File.Open() FileInfo.Open()	Δημιουργεί ένα FileStream χρησιμοποιώντας FileMode και FileAccces. Ισοδύναμο με τον FileStream Constructor
File.OpenRead() FileInfo.OpenRead()	Ανοίγει ένα ρεύμα μόνο για ανάγνωση. Ισοδύναμεί με τον καθορισμό των ορισμάτων FileMode.Open και FileAccess.Read
File.OpenText() FileInfo.OpenText() File.AppendText() FileInfo.AppendText() File.CreateText() FileInfo.CreateText()	Δημιουργεί έναν αναγνώστη ρεύματος (StreamReader)
File.OpenWrite() FileInfo.OpenWrite()	Ανοίγει ένα ρεύμα μόνο για ανάγνωση. Ισοδύναμο με τη χρήση των ορισμάτων FileMode.Open και FileAccess.Write

Αν και η δομή της κλάσης είναι διαφορετική, τα περισσότερα από τα χαρακτηριστικά των αρχείων συστήματος από την κλάση **java.io.File** μπορούν να βρεθούν στο **System.IO namespace** της C#. Ο πίνακας που ακολουθεί παρέχει την αντιστοιχία μεταξύ των κλάσεων της Java και της C# όπου μια μέθοδος που ομαδοποιείται στις κλάσεις File και Directory, έχει μια αντίστοιχη μέθοδο στις κλάσεις FileInfo και DirectoryInfo αντίστοιχα του .NET

JAVA	**.NET**
File(StringPath)	FileInfo(path) DirInfo(path)
canRead()	N/A
canWtite()	N/A
compareTo()	N/A
createNewFile()	File.Create() Directory.Create()
createTempFile()	Path.GetTempFileName()
Delete()	File.Delete() Directory.Delete()
deleteOnExit()	N/A
exists()	File.Exists() Directory.Exists()
getAbsolutePath() getAbsoluteName()	N/A
getCanonicalPath() getCanonicalName()	Path.GetFullPath()
getName()	Path.GetFileName()
getParent()	Path.GetDirectoryName() Directory.GetParent()
isAbsolute()	Path.IsPathRooted()
isFile()	File.Exists()
isDirectory()	Directory.Exists()

isHidden()	File.GetAttributes().Hiddden Directory.GetAttributes().Hiddden
lastModified()	File.GetLastWriteTime() Directory.GetLastWriteTime()
Length()	FileInfo.Length
listFiles()	Directory.GetFileSystemEntries() Directory.GetFiles() Directory.GetDirectories()
listRoots()	Directory.GetLogicalDrives()
mkdir() mkdirs()	Directory.CreateDirectory()
renameTo()	Directory.Move() File.Move()
setLastModified()	File.SetLastWriteTime() Directory.SetLastWriteTime()
isReadOnly()	File.Attributes.ReadOnly
N/A	File.Copy
N/A	File.GetAttributes Directory.GetAttributes
N/A	File.SetAttributes Directory.SetAttributes
N/A	File.SetCreationTime() File.GetCreationTime() File.SetLastAccessTime() File.GetLastAccessTime() Directory.SetCreationTime() Directory.GetCreationTime() Directory.SetLastAccessTime() Directory.GetLastAccessTime()
N/A	Directory.GetCurrentDirectory() Directory.SetCurrentDirectory()

9.9 ΚΑΤΕΥΘΥΝΟΜΕΝΗ ΔΙΑΔΙΚΑΣΙΑ

Οτιδήποτε μέχρι στιγμής έχει αναλυθεί έχει σχέση με την σειριακή προσπέλαση αρχείων σε CS ή BS ρεύματα, όπου ο **εσωτερικός δείκτης προσπέλασης του αρχείου (File Access Pointer)** μετακινείται σειριακά από τον ένα χαρακτήρα στον άλλο. Ωστόσο, η πλέον χρήσιμη προσπέλαση είναι η **τυχαία προσπέλαση (Random Access)** και μάλιστα η ειδική μορφή αυτής η **κατευθυνόμενη προσπέλαση (Direct Access)** σε BS ρεύματα.

Στην τυχαία προσπέλαση, ο χρήστης προσδιορίζει το σημείο που θα βρεθεί ο εσωτερικός δείκτης προσπέλασης του αρχείου, είτε για ανάγνωση, είτε για εγγραφή. Αν ο προσδιορισμός αυτός δίνεται με την τιμή του αύξοντος αριθμού του byte, έχουμε την τεχνική του **File Position.** Ωστόσο, υπάρχουν περιπτώσεις που η δομή του αρχείου μας είναι σε εγγραφές (records) δηλαδή, σε αντικείμενα και μάλιστα σε τιμές σταθερού μήκους. Σε τέτοιες περιπτώσεις το σημείο προσπέλασης του δείκτη δίνεται με **σταθερό αριθμό εγγραφής (Record Number)** από το χρήστη, οπότε έχουμε την ειδική περίπτωση του **Direct Access.** Η κατευθυνόμενη διαδικασία είναι πραγματικά χρήσιμη, σε αρχεία που έχουν σταθερό μήκος οι εγγραφές τους είτε αυτές είναι δομημένες σε records είτε όχι. Αντίθετα η τεχνική του File Position είναι χρήσιμη σε

γενικευμένες περιπτώσεις που το μήκος κάθε εγγραφής δεν είναι σταθερό, οπότε το σημείο που αρχίζει η κάθε μια πρέπει να προσδιοριστεί ή να αναγνώθει από κάποιο βοηθητικό αρχείο που περιέχει αυτές τις θέσεις. Γλώσσες όπως οι Fortran, Cobol και Pascal λειτουργούσαν με κατευθυνόμενη διαδικασία ενώ οι νεότερες όπως οι C#, Java, C++, λειτουργούν με τυχαία προσπέλαση. Ευτυχώς οι τελευταίες δεν έχουν πρόβλημα διότι ισχύει το ακόλουθο θεώρημα επικοινωνίας των δύο τεχνικών :

ΘΕΩΡΗΜΑ ΤΥΧΑΙΑΣ ΠΡΟΣΠΕΛΑΣΗΣ

Αν **rec** είναι ο **αριθμός εγγραφής** και **sizerec μήκος κάθε εγγραφής,** τότε η πληροφορία του αντίστοιχου record θα αρχίζει από τη θέση **pos** σε σχέση με την αρχή του αρχείου:

pos=(rec-1)*(sizerec+2) (1)

pos=(rec-1)*(sizerec) (2)

όπου ην σχέση (1) ισχύει για **αρχεία κειμένου** και η σχέση (2) για **binary αρχεία.** Η αναζήτηση μέσα σε ένα αρχείο γίνεται με την ειδική μέθοδο **Seek** η οποία έχει συντακτικό τύπο:

public override long Seek(long offset, SeekOrigin origin);

Όπου offset καθορίζει από που θα ξεκινήσει η αναζήτηση και το origin καθορίζει την αρχή, το τέλος ή την τρέχουσα θέση σαν σημείο αναφοράς. Η μέθοδος αυτή επιστρέφει τιμή την νέα θέση στο ρεύμα.

9.10 SERIALIZATION

Από τις σημαντικότερες δυνατότητες του αντικειμενοστρεφούς προγραμματισμού είναι η δυνατότητα I/O σε **ρεύματα των bytes (SB)** σε επίπεδο αντικείμένων και βέβαια, η αυτόματη καταχώρηση και ανάκτηση των δεδομένων αυτών από αρχεία. Η εργασία καταχώρησης ονομάζεται: **Serialization** και η ανάκτηση **Deserialization.**

Ο αυτοματισμός I/O με αντικείμενα έχει νόημα σε πολλές περιπτώσεις στην πράξη. Μια βασική εφαρμογή είναι η τοπική πληκτρολόγηση εγγράφων από διάφορους χρήστες και στη συνέχεια η αυτόματη μαζική μεταφορά αυτών και καταχώριση στη βάση δεδομένων. Βέβαια, μια τέτοια δυνατότητα προϋποθέτει πως και η καταχώριση και η ανάκτηση, επιτρέπει **μεταβλητού μήκους εγγραφές και αντικείμενα που μπορεί να αναφέρονται σε άλλα.**

Στη Java, υπάρχει η διαπροσωπεία **Serializable** με κενό περιεχόμενο από μέλη. Από την άλλη μεριά το .NET χρησιμοποιεί τη διαπροσωπεία **System.Runtime.Serialization.IFormatters** η οποία χρησιμοποιεί δύο εφαρμογές τις:

1. System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
2. System.Runtime.Serialization.Formatters.SOAP.SoapFormatter

Στη Java οι κλάσεις δεν είναι serializable εξ' ορισμού. Στο .NET απαιτείται η χρήση της δήλωσης **[Serializable]** πριν από την κλάση.

Παράλληλα μας δίνεται η δυνατότητα να επιλέξουμε πότε θέλουμε να εφαρμόσουμε την δυνατότητα serializable και πότε όχι. Αυτό γίνεται με τη χρήση της ειδικής δήλωσης **[NonSerialized]** και αυτό φαίνεται ξεκάθαρα στον παρακάτω κώδικα:

```
[Serializable]
class example{
[NonSerialized] private int o_int;
private string o_string;}
```

9.11 ΑΣΚΗΣΕΙΣ

1. Να μετατραπεί ο παρακάτω κώδικας Java σε κώδικα C#

```
/* Copy a file to a destination file
   and see the contents of it.    */
import java.io.*;
class Java052
{ public static void main(String x[])
  throws IOException
  {  PrintStream out = System.out;
     String name_of_source;
     String name_of_destination;
     Console object = new Console();
     boolean criterion;
     FileInputStream filein;
     FileOutputStream fileout;
     int intbyte;
     filein  = null;
     fileout = null;
     do
     { criterion = false;
       out.print("Full name of source file ==>");
       name_of_source = object.console_string();
       try
       {filein=new FileInputStream(name_of_source);}
       catch(FileNotFoundException e)
       { out.println("Source file not found...");
         criterion=true;}}
     while(criterion);
     do
     { criterion = false;
       out.print("Full name of destination file ==>");
       name_of_destination = object.console_string();
       try
       {fileout = new FileOutputStream(name_of_destination);}
       catch(FileNotFoundException e)
       { out.println("Error in opening output file!");
         criterion=true;}}
     while(criterion);
     try{
    do
       {  intbyte = filein.read();
           if(intbyte != -1)
           {  fileout.write(intbyte);
              out.print((char)intbyte);}}
      while(intbyte != -1);}
     catch(IOException e)
```

```
   {  out.println("File error..."); }
   filein.close();
   fileout.close(); }}
```

2. Να μετατραπεί ο παρακάτω κώδικας Java σε κώδικα C#

```
/* Serialization : store objects in
   a file and use them later...  */
import java.io.*;
class Medicin implements Serializable
{  String  name;
   int    stock;
   int     safe;
   double  cost;
   public Medicin(String n,int st, int sa,double c)
   {  name = n;
      stock= st;
      safe = sa;
      cost = c;
   }
   public String toString()
   {  return name+' '+stock+' '+safe+' '+cost;
   }
}
class Java050{
  public static void main(String x[])
  throws IOException
  {  PrintStream out = System.out;
     boolean criterion = true;
     String name0;
     int stock0,safe0;
     double cost0;
     Medicin object;
     // input file
     Console in = new Console();
     out.print("Full name of DBfile ==>");
     String f = in.console_string();
     FileOutputStream  datafile = new FileOutputStream(f);
     ObjectOutputStream file =new ObjectOutputStream(datafile);
     // input some records
     while(criterion)
     {  object=null;
        out.print("Name(STOP for end)==>");
        name0=in.console_string();
        if(name0.equals("STOP"))
           criterion=false;
        else
        {  out.print(" Stock (int) ==>");
           stock0=in.console_int();
```

```
                out.print(" Safe (int)  ==>");
                safe0=in.console_int();
                out.print(" Cost(double)==>");
                cost0=in.console_double();
                object = new Medicin(name0,stock0,safe0,cost0);
                file.writeObject(object);}}
          file.flush();
          file.close();
          // read data
          FileInputStream  readdata =new FileInputStream(f);
          ObjectInputStream infile = new
        ObjectInputStream(readdata);
          Medicin inobject;
          out.println("YOU GAVE THE FOLLOWING DATA:");
          do
          {  inobject=null;
             try
             { inobject = (Medicin) infile.readObject();}
             catch(Exception e){}
             if(inobject == null) break;
             out.println(inobject);}
          while(true);
          infile.close();
    }
    }
```

3. Στο ASCII αρχείο emp.dat έχουν καταγραφεί οι υπάλληλοι εταιρείας όπου κάθε γραμμή περιλαμβάνει τα ακόλουθα στοιχεία:

Όνομα	(60 χαρακτήρες)
Διεύθυνση	(100 χαρακτήρες)
Ειδικότητα	(50 χαρακτήρες)
Τηλέφωνο	(10 χαρακτήρες)
Μικτός Μισθός	(8 χαρακτήρες)

όπου ο κωδικός αριθμός κάθε ατόμου ταυτίζεται με τον κωδικό γραμμής των στοιχείων του (πχ ο υπάλληλος 10 έχει τα υπόλοιπα στοιχεία του στη 10η γραμμή).

Α) Να γράφει πρόγραμμα C ASCII αρχείου αναφοράς report.dat όπου κάθε γραμμή να περιέχει τα στοιχεία του υπάλληλου. Το report.dat περιέχει μόνο εκείνους τους υπαλλήλους που ικανοποιούν τα ακόλουθα στοιχεία:

Εχουν κωδικό από i1 μέχρι i2 και

Μικτό Μισθό μικρότερο των 1000 €.

Οι παράμετροι i1,i2 δίνονται κατά την διάρκεια εκτέλεσης του προγράμματος.

Β) Στο ASCII αρχείο kratisis.txt είναι αποθηκευμένα τα ακόλουθα στοιχεία:

Επίδομα τέκνου	(8 χαρακτήρες)
Αριθμός των τέκνων κάθε υπαλλήλου	(1 χαρακτήρας)

όπου ο κωδικός γραμμής κάθε ατόμου ταυτίζεται με τον κωδικό γραμμής των στοιχείων του (πχ η γραμμή 10 του πρώτου αρχείου έχει τα υπόλοιπα στοιχεία της στην 10η γραμμή του δεύτερου αρχείου και αντιστοιχεί στον υπάλληλο με κωδικό 10).

Να γράφει πρόγραμμα C ASCII αρχείου αναφοράς report.dat όπου κάθε γραμμή να περιέχει τα στοιχεία του υπάλληλου και το μισθό που του αναλογεί μετά από τις κρατήσεις και τα επιδόματα. Το report.dat περιέχει μόνο εκείνους τους υπάλληλους που ικανοποιούν τα ακόλουθα στοιχεία:

Ο καθαρός μισθός ενός υπάλληλου είναι

Μισθός = Μικτός_Μισθός-(5%ΙΚΑ * Μικτός_Μισθός + 8%φορος * Μικτός_ Μισθός) + επίδομα_ τέκνων * τέκνα

Το report.dat περιέχει μόνο εκείνους τους υπαλλήλους που ικανοποιούν τα ακόλουθα στοιχεία:
Εχουν κωδικό από i1 μέχρι i2
και Μικτό Μισθό μικρότερο των 1000 €
και 2 παιδιά.

ΚΕΦΑΛΑΙΟ 10ο

10. ΠΑΡΑΘΥΡΙΚΕΣ ΕΦΑΡΜΟΓΕΣ

10.1 ΕΙΣΑΓΩΓΗ

Ο χρήστης της Java, γνωρίζει ήδη ότι όταν θέλει να σχεδιάσει και να αναπτύξει γραφικά και παραθυρικές εφαρμογές, αυτό μπορεί να το κατορθώσει με ειδικές κλάσεις μέσα από τα **Swing/GFC, awt, java2D** για μια γρήγορη ανάπτυξη. Στη C# οι ίδιες δυνατότητες και περισσότερες, επιτυγχάνονται μέσα από τις βιβλιοθήκες του .NET. Συγκεκριμένα, από το **VISUAL STUDIO.NET** μπορεί να εκμεταλλευτεί κανείς το **IDE (Integrated Development Environment)** για να σχεδιάσει και να ενσωματώσει στον κώδικα αυτόματα κάποιο γεγονός (αντικείμενο) παραθυρικής μορφής. Συγκεκριμένα, αν κάποιος γνωρίζει το χειρισμό του VISUAL STUDIO για VISUAL C++ ή και για VISUAL BASIC, ακόμα μπορεί με τον ίδιο τρόπο να χρησιμοποιήσει τις ίδιες δυνατότητες και στη C#. Βεβαίως, υπάρχουν περιπτώσεις που κάτι παρέχεται για VISUAL BASIC ή για C++ και δεν παρέχεται στη C#.

Δεν πρέπει ο χρήστης να νομίσει ότι είναι αναγκαστική η ύπαρξη λειτουργίας του VISUAL STUDIO.NET για να εργαστεί με C#. Ακόμη και σε περίπτωση χρήσης windows μέσα στην εφαρμογή του, το μόνο που χρειάζεται είναι ένας **text editor** και ένας compiler της C# ακόμη και σε command line επίπεδο. Βέβαια, αναπτύσσοντας μια εφαρμογή μέσα από το VISUAL STUDIO.NET που είναι **GUI** περιβάλλον, είναι ευνόητο ότι η αυτόματη παραθυρική εισαγωγή κώδικα είναι ευκολότερη.

Επομένως, δεν έχει μεγάλη σημασία η σύγκριση στην περίπτωση αυτή μεταξύ Java και C# των υπαρχόντων κλάσεων, ωστόσο, για να γνωρίζει ο χρήστης της Java απλά και μόνο σε υπάρχοντα προγράμματα προς τα που πρέπει να στραφεί για την πλευρά της C#, παραθέτουμε τους πίνακες που ακολουθούν για την αντιστοίχηση κλάσεων και μεθόδων

JAVA	.**NET**
AlphaComposite	N/A
AWTEvent	System.EventArgs
AWTEventMulticaster	N/A
AWTKeyStroke	System.Windows.Forms.KeyPressEventArgs
AWTPermition	System.Security.Permissions.UIPermission
BasicStroke	System.Drawing.Pen
BorderLayout	N/A
BufferCapabilities	N/A
Button	System.Windows.Forms.Button
Canvas	System.Windows.Forms.Control
CardLayout	N/A
Checkbox	System.Windows.Forms.RadioButton
CheckboxGroup	N/A
CheckboxMenuItem	N/A
Choice	System.Windows.Forms.ComboBox

Color	System.Drawing.Color
Component	System.Windows.Forms.Control
ComponentOrientation	N/A
Container	System.Windows.Forms.Control
Cursor	System.Windows.Forms.Cursor
Dialog	System.Windows.Forms.CommonDialog
Dimension	System.Drawing.Size
DisplayMode	N/A
Event	System.EventArgs
EventQueue	N/A
FileDialog	System.Windows.Forms.FileDialog System.Windows.Forms.OpenFileDialog System.Windows.Forms.SaveFileDialog
FlowLayout	N/A
FocusTraversalPolicy	N/A
Font	System.Drawing.Font
FontMetrics	N/A
Frame	System.Windows.Forms.Form
GradientPaint	System.Drawing.Drawing2D.LinearGradientBrush
Graphics	System.Drawing.Graphics
Graphics2D	System.Drawing.Graphics
GraphicsConfigTemplate	N/A
GraphicsConfiguration	N/A
GraphicsDevice	N/A
GraphicsEnvironment	N/A
GridBagConstraints	N/A
GridBagLayout	N/A
GridLayout	N/A
Image	System.Drawing.Image
ImageCapabilities	N/A
Insets	N/A
JobAttributes	System.Drawing.Printing.PageSettings System.Drawing.Printing.PrinterSettings
KeyboardFocusManager	N/A
Label	System.Windows.Forms.Label
List	System.Windows.Forms.ListBox
MediaTracker	N/A
Menu	System.Windows.Forms.MainMenu
MenuBar	N/A
MenuComponent	N/A
MenuItem	System.Windows.Forms.MenuItem
MenuShortcut	N/A
PageAttributes	System.Drawing.Printing.PageSettings
Panel	System.Windows.Forms.Panel
Point	System.Drawing.Point
Polygon	N/A
PopupMenu	System.Windows.Forms.ContextMenu
PrintJob	System.Drawing.Printing.PrintDocument
Rectangle	System.Drawing.Rectangle
RenderingHints	N/A
RenderingHints.Key	N/A

Robot	N/A
Scrollbar	System.Windows.Forms.HScrollBar System.Windows.Forms.VscrollBar
ScrollPane	System.Windows.Forms.Panel
ScrollPaneAdjustable	N/A
SystemColor	N/A
TextArea	System.Windows.Forms.TextBox System.Windows.Forms.RichTextBox
TextComponent	N/A
TextField	System.Windows.Forms.TextBox
TexturePaint	System.Drawing.TextureBrush
Toolkit	N/A
Window	N/A
ActionEvent	System.EventArgs
AdjustmentEvent	System.Windows.Forms.ScrollEventArgs
AWTEventListenerProxy	N/A
ComponentAdapter	N/A
ComponentEvent	System.EventArgs
ContainerAdapter	N/A
ContainerEvent	System.Windows.Forms.ControlEventArgs
FocusAdapter	N/A
FocusEvent	System.EventArgs
HierarchyBoundsAdapter	N/A
HierarchyEvent	N/A
InputEvent	N/A
InputMethodEvent	N/A
InvocationEvent	N/A
ItemEvent	System.EventArgs
KeyAdapter	N/A
KeyEvent	System.Windows.Forms.KeyPressEventArgs System.Windows.Forms.KeyEventArgs
MouseAdapter	N/A
MouseEvent	System.Windows.Forms.MouseEventArgs
MouseMotionAdapter	N/A
MouseWheelEvent	System.Windows.Forms.MouseEventArgs
PaintEvent	System.Windows.Forms.PaintEventArgs
TextEvent	System.EventArgs
WindowAdapter	N/A
WindowEvent	System.EventArgs
Book	N/A
PageFormat	System.Drawing.Printing.PageSettings
Paper	System.Drawing.Printing.PaperSize
PrinterJob	System.Drawing.Printing.PrintDocument

10.2 ΣΧΕΔΙΑΣΜΟΣ

Κανονικά όλες οι κλάσεις που απαιτούνται για σχεδιασμό και υλοποίηση εφαρμογών που υπήρχαν στο **Microsoft Foundation Classes (MFC)** του VISUAL STUDIO τώρα στην C# υπάρχουν στο **System.Windows.Forms.** Βέβαια μπορεί κάποιος να χρησιμοποιήσει ακόμα για να ζωγραφίσει γραμμές

για παράδειγμα το **System.Drawing**, καθώς επίσης κλάσεις για ειδικά fonts και τυποποιημένα αντικείμενα.

Θα συνεχίσουμε τώρα με την δημιουργία μιας παραθυρικής εφαρμογής ξεκινώντας το χτίσιμο βήμα προς βήμα. Αρχικά θα δημιουργήσουμε μια φόρμα μέσα από **command line**. Ανοίξτε το **notepad** και πληκτρολογείστε τον παρακάτω κώδικα:

```
using System;
using System.Windows.Forms;

public class NForm:Form{
  public NForm(){}
public static void Main(string[] args){
NForm cform=new NForm();
Application.Run(cform);}}
```

Σώστε το αρχείο με ότι όνομα θέλετε και στην συνέχεια κάντε το compile με ένα compiler της C# :

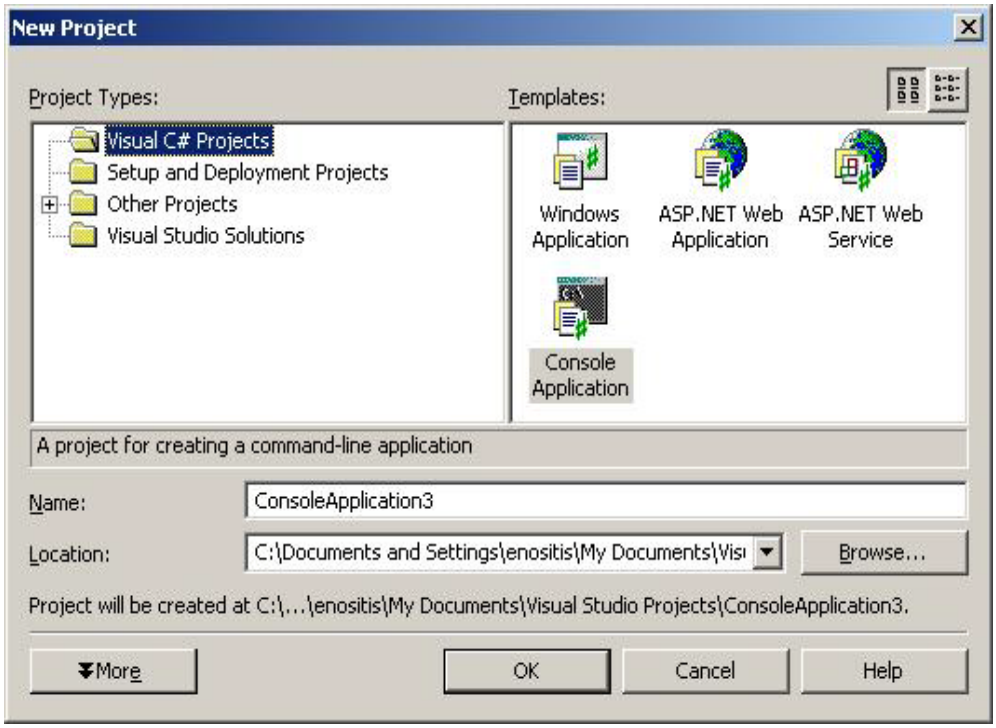
csc testform.cs

Τότε δημιουργείται η ακόλουθη φόρμα

Ιδιαίτερη προσοχή θα πρέπει να προσέξουμε στο εξής γεγονός. Αν χρησιμοποιούμε εφαρμογές τύπου console application και προσθέσουμε την παρακάτω γραμμή **using System.Windows.Forms;** θα παρουσιαστεί πρόβλημα διότι δεν αναγνωρίζει ο κώδικας την βιβλιοθήκη αυτή. Απαιτειται λοιπον η καταλληλη παραμετρος η το reference στην βιβλιοθηκη μεσα αποτο περιβαλλον VISUAL STUDIO.NET

Αξιζει να προχωρήσουμε ένα βήμα παραπάνω και να δημιουργήσουμε την πρώτη ολοκληρωμένη, απλή, παραθυρική εφαρμογή. Αρχικά ανοίγουμε το **MS VISUAL STUDIO.NET** . Από το menu **FILE➔New➔Project** ξεκινάμε μια

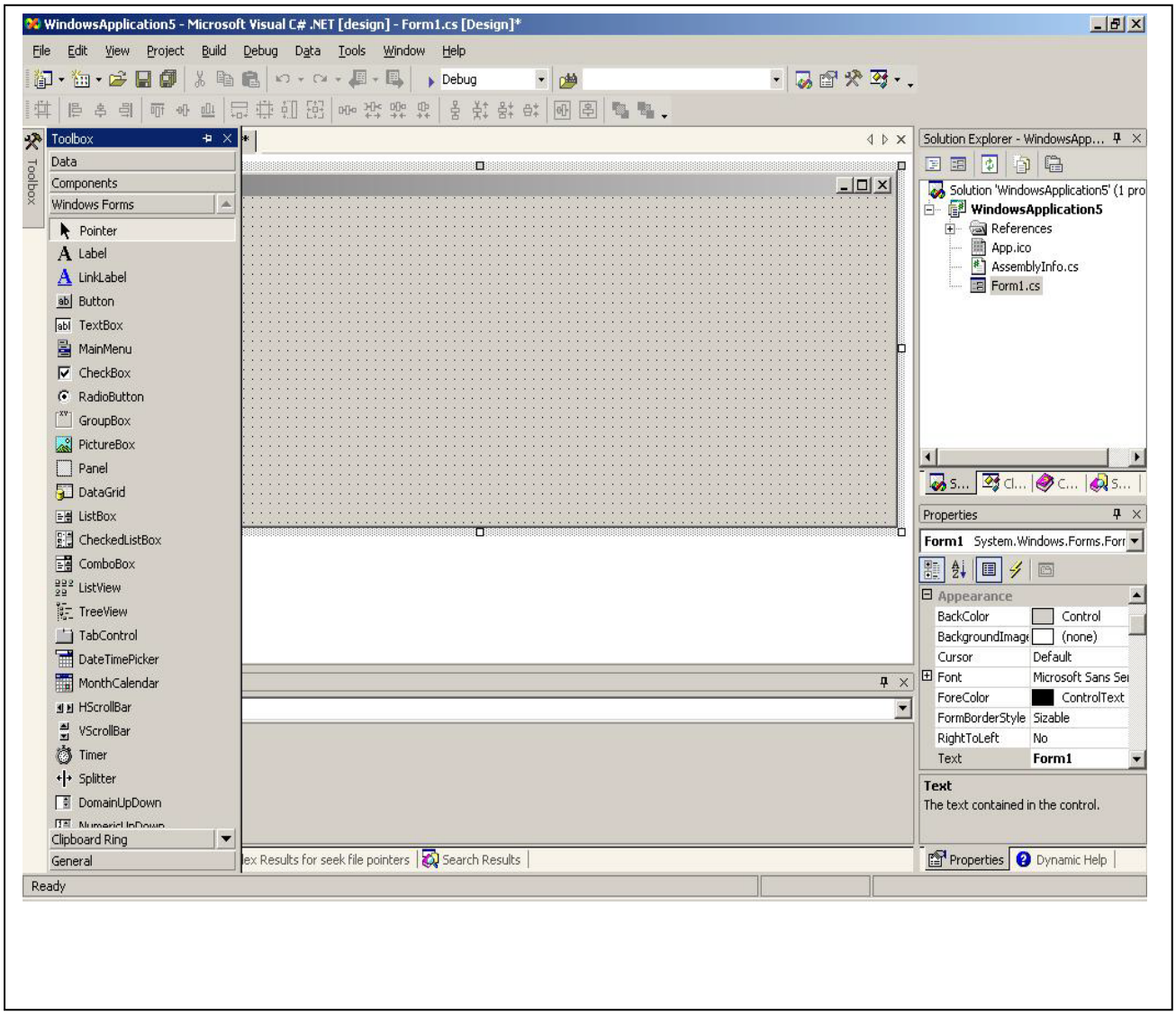
νέα εφαρμογή. Επιλέγουμε από την κάτωθι οθόνη το είδος της εφαρμογής, δίνουμε ένα όνομα και την σώζουμε στο φάκελο που επιθυμούμε. Για την συγκεκριμένη εφαρμογή θα δημιουργήσουμε μια **Windows Application** .



Το επόμενο βήμα είναι η εμφάνιση στην κεντρική οθόνη όπου εκεί θα χτιστεί η εφαρμογή μας. Στο κέντρο της οθόνης είναι η φόρμα πάνω στην οποία θα τοποθετήσουμε τα Component για την εφαρμογή μας. Στα αριστερά της οθόνης εμφανίζεται μια λίστα επιλογής αντικειμένων που μπορούμε να χρησιμοποιήσουμε στην εφαρμογή μας. Χρήσιμα εργαλεία για την δημιουργία παραθυρικών εφαρμογών είναι αντικείμενα της βιβλιοθήκης **System.Windows.Forms.Control.** Controls όπως:

- **Label**
- **Buttons**
- **Checkboxes**
- **Menus**
- **Listboxes**
- **Radiobuttons**
- **Textboxes**
- **Tool bars**
- **Tree view**

Στα δεξιά της οθόνης βρίσκεται το παράθυρο των **Properties** της φόρμας. Εκεί ρυθμίζονται όλες οι επιλογές και οι ιδιότητες των αντικειμένων που έχουν επιλεχθεί. Όλα τα παραπάνω φαίνονται στην παρακάτω εικόνα:



Ειδικότερα ο κώδικας για την κατασκευή της απλής αυτής φόρμας φαίνεται παρακάτω. Δώστε ιδιαίτερη προσοχή στις διάφορες του κώδικα με τον αντίστοιχο του απλού παραδείγματος που παρουσιάσαμε παραπάνω.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace WindowsApplication1{
	/// <summary>
	/// Summary description for Form1.
	/// </summary>
	public class Form1 : System.Windows.Forms.Form{
		/// <summary>
		/// Required designer variable.
		/// </summary>
	private System.ComponentModel.Container components = null;
```

```
            public Form1(){
                  //
                  // Required for Windows Form Designer support
                  //
                  InitializeComponent();

                  //
                  // TODO: Add any constructor code after
InitializeComponent call
                  //
            }

            /// <summary>
            /// Clean up any resources being used.
            /// </summary>
            protected override void Dispose( bool disposing ){
                  if( disposing ){
                        if (components != null){
                              components.Dispose();}}
                  base.Dispose( disposing );}

            #region Windows Form Designer generated code
            /// <summary>
            /// Required method for Designer support - do not modify
            /// the contents of this method with the code editor.
            /// </summary>
            private void InitializeComponent(){
            this.components = new System.ComponentModel.Container();
                  this.Size = new System.Drawing.Size(300,300);
                  this.Text = "Form1";}
            #endregion

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

Η εφαρμογή την οποία θα σχεδιάσουμε είναι ένα απλό κομπιουτεράκι. Ο χρήστης θα δίνει δυο ακεραίους, θα επιλέγει το είδος της πράξης και στην συνέχεια πατώντας το κουμπί θα εμφανίζεται το αποτέλεσμα σε μια νέα φόρμα. Αρχικά προσθέτουμε τα δυο label με ονόματα ΑΚΕΡΑΙΟΣ1, ΑΚΕΡΑΙΟΣ2. Τα label είναι κείμενα που εμφανίζονται σε μια παραθυρική εφαρμογή. Ο κώδικας παίρνει την μορφή:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace WindowsApplication1{
      /// <summary>
      /// Summary description for Form1.
      /// </summary>
      public class Form1 : System.Windows.Forms.Form{
/*Προσέξτε τον κώδικα που προστίθενται. Ιδιαίτερα πως ορίζονται τα label στον constructor της φόρμας.*/
```

```
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;

    private System.ComponentModel.Container components = null;
        public Form1(){
            InitializeComponent();
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing){
            if( disposing ){
                if (components != null){
                    components.Dispose();}}
            base.Dispose( disposing );}

        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent(){
```

/* Υλοποιούμε τα label και δεσμεύουμε τα απαραίτητα διαμερίσματα μνημης.*/

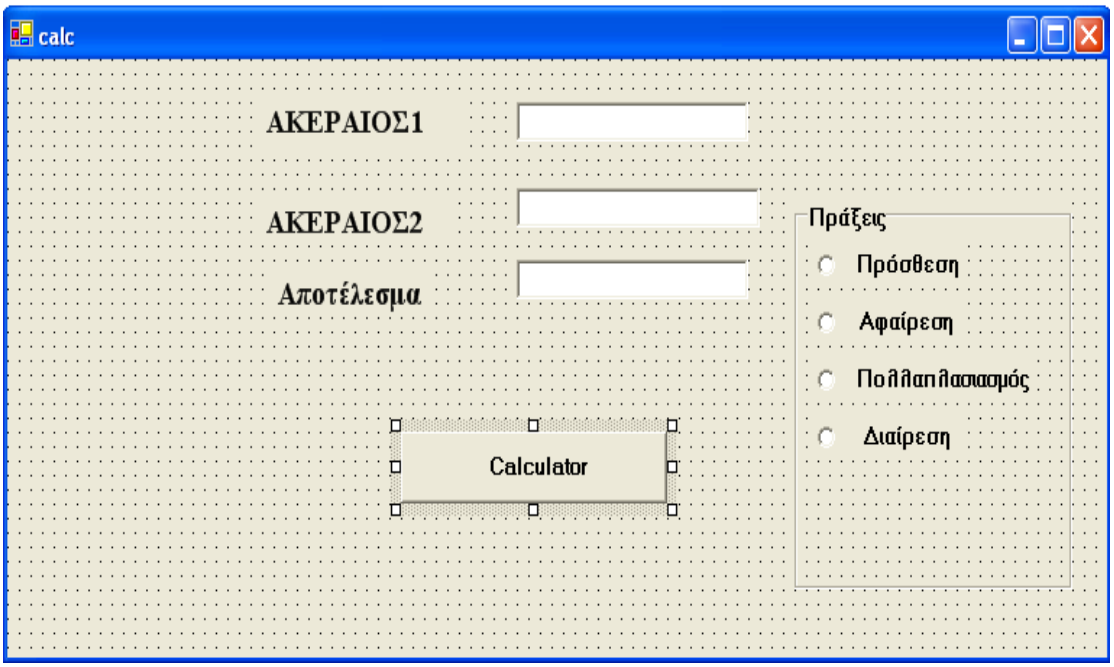
```
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.SuspendLayout();
```

/*Ενώ μέσω του παραθύρου Properties μπορούμε με απλά κλικ στο mouse να ορίσουμε τιμές,fonts,μέγεθος, συντεταγμένες στην οθόνη παρακάτω φαίνεται ο κώδικας που απαιτείται για αντίστοιχη υλοποίηση*/

```
this.label1.Font = new System.Drawing.Font("Times New Roman", 14.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(161)));
this.label1.Location = new System.Drawing.Point(176, 24);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(144, 32);
this.label1.TabIndex = 0;
this.label1.Text = "ΑΚΕΡΑΙΟΣ1";
this.label2.Font = new System.Drawing.Font("Times New Roman", 14.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(161)));
this.label2.Location = new System.Drawing.Point(176, 80);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(136, 24);
this.label2.TabIndex = 1;
this.label2.Text = "ΑΚΕΡΑΙΟΣ2";
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.ClientSize = new System.Drawing.Size(760, 334);
this.Controls.AddRange(new
System.Windows.Forms.Control[]{this.label2,this.label1});
this.Name = "Form1";
this.Text = "calc";
this.ResumeLayout(false);     }
#endregion
```

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

Προσθέτουμε στην φόρμα όλα τα στοιχεία για να είναι έτοιμο ένα μέρος της παραθυρικής εφαρμογής μας. Αυτό φαίνεται στην εικόνα που ακολουθεί ενώ στη συνέχεια παρατίθεται και ο κώδικας υλοποίησης του προγράμματος μας.



Στις δύο παρακάτω συναρτήσεις, βλέπουμε τη μορφή που έχει πάρει ο κώδικας, όλα τα αντικείμενα που έχουν προστεθεί πάνω στη φόρμα, καθώς επίσης τιμές και ιδιότητες των properties αυτών.

```
public class Form1 : System.Windows.Forms.Form
      {
            private System.Windows.Forms.Label label1;
            private System.Windows.Forms.Label label2;
            private System.Windows.Forms.TextBox textBox1;
            private System.Windows.Forms.TextBox textBox2;
            private System.Windows.Forms.Button button1;
            private System.Windows.Forms.RadioButton radioButton1;
            private System.Windows.Forms.GroupBox groupBox1;
            private System.Windows.Forms.RadioButton radioButton2;
            private System.Windows.Forms.RadioButton radioButton3;
            private System.Windows.Forms.RadioButton radioButton4;
            private void InitializeComponent()
            {
                  this.label1 = new System.Windows.Forms.Label();
                  this.label2 = new System.Windows.Forms.Label();
                  this.textBox1 = new System.Windows.Forms.TextBox();
                  this.textBox2 = new System.Windows.Forms.TextBox();
this.button1 = new System.Windows.Forms.Button();
this.radioButton1 = new System.Windows.Forms.RadioButton();
```

```
this.groupBox1 = new System.Windows.Forms.GroupBox();
this.radioButton2 = new System.Windows.Forms.RadioButton();
this.radioButton3 = new System.Windows.Forms.RadioButton();
this.radioButton4 = new System.Windows.Forms.RadioButton();
this.groupBox1.SuspendLayout();
this.SuspendLayout();
this.label1.Font = new System.Drawing.Font("Times New Roman", 14.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(161)));
this.label1.Location = new System.Drawing.Point(176, 24);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(144, 32);
this.label1.TabIndex = 0;
this.label1.Text = "ΑΚΕΡΑΙΟΣ1";
this.label2.Font = new System.Drawing.Font("Times New Roman", 14.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(161)));
this.label2.Location = new System.Drawing.Point(176, 80);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(136, 24);
this.label2.TabIndex = 1;
this.label2.Text = "ΑΚΕΡΑΙΟΣ2";
this.textBox1.Font = new System.Drawing.Font("Microsoft Sans Serif",
9.75F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((System.Byte)(161)));
this.textBox1.ForeColor = System.Drawing.Color.DeepPink;
this.textBox1.Location = new System.Drawing.Point(352, 24);
this.textBox1.Name = "textBox1";
this.textBox1.Size = new System.Drawing.Size(160, 22);
this.textBox1.TabIndex = 2;
this.textBox1.Text = "";
this.textBox2.ForeColor = System.Drawing.Color.Coral;
this.textBox2.Location = new System.Drawing.Point(352, 72);
this.textBox2.Name = "textBox2";
this.textBox2.Size = new System.Drawing.Size(168, 20);
this.textBox2.TabIndex = 3;
this.textBox2.Text = "";
this.button1.Font = new System.Drawing.Font("Microsoft Sans Serif",
9.75F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((System.Byte)(161)));
this.button1.Location = new System.Drawing.Point(272, 208);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(184, 40);
this.button1.TabIndex = 4;
this.button1.Text = "Calculator";
this.radioButton1.Location = new System.Drawing.Point(16, 24);
this.radioButton1.Name = "radioButton1";
this.radioButton1.TabIndex = 5;
this.radioButton1.Text = "Πρόσθεση";
this.radioButton1.TextAlign =
System.Drawing.ContentAlignment.TopCenter;
this.groupBox1.Controls.AddRange(new System.Windows.Forms.Control[]
{this.radioButton4,this.radioButton3,this.radioButton2,this.radioButt
on1});
this.groupBox1.Font = new System.Drawing.Font("Microsoft Sans Serif",
9.75F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((System.Byte)(161)));
this.groupBox1.Location = new System.Drawing.Point(544, 80);
this.groupBox1.Name = "groupBox1";
this.groupBox1.Size = new System.Drawing.Size(192, 216);
this.groupBox1.TabIndex = 6;
```

```
this.groupBox1.TabStop = false;
this.groupBox1.Text = "Πράξεις";
this.radioButton2.Location = new System.Drawing.Point(16, 56);
this.radioButton2.Name = "radioButton2";
this.radioButton2.TabIndex = 6;
this.radioButton2.Text = "Αφαίρεση";
this.radioButton2.TextAlign =
System.Drawing.ContentAlignment.TopCenter;
this.radioButton3.Location = new System.Drawing.Point(16, 88);
this.radioButton3.Name = "radioButton3";
this.radioButton3.Size = new System.Drawing.Size(152, 24);
this.radioButton3.TabIndex = 7;
this.radioButton3.Text = "Πολλαπλασιασμός";
this.radioButton3.TextAlign =
System.Drawing.ContentAlignment.TopCenter;
this.radioButton4.Location = new System.Drawing.Point(16, 120);
this.radioButton4.Name = "radioButton4";
this.radioButton4.TabIndex = 8;
this.radioButton4.Text = "Διαίρεση";
this.radioButton4.TextAlign =
System.Drawing.ContentAlignment.TopCenter;
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.ClientSize = new System.Drawing.Size(760, 334);
this.Controls.AddRange(new System.Windows.Forms.Control[]
{this.groupBox1,  this.button1, this.textBox2,  this.textBox1,
this.label2,  this.label1});
this.Name = "Form1";
this.Text = "calc";
this.groupBox1.ResumeLayout(false);
this.ResumeLayout(false);     }
```

Κάνοντας διπλό κλικ στο κουμπί της φόρμας και αφού έχουμε τοποθετήσει τις τιμές στα αντίστοιχα textboxes εμφανίζεται το επιθυμητό αποτέλεσμα. Ο κώδικας του αντιστοίχου κουμπιού είναι ο κάτωθι:

```
if (radioButton1.Checked){
      double j= double.Parse(textBox1.Text);
      double k=double.Parse(textBox2.Text);
      double m= j+k;
      textBox3.Text= m.ToString();}
      if (radioButton2.Checked){
            double j= double.Parse(textBox1.Text);
                  double k=double.Parse(textBox2.Text);
                  double m= j-k;
                  textBox3.Text= m.ToString();}
                  if (radioButton3.Checked){
                        double j= double.Parse(textBox1.Text);
                        double k=double.Parse(textBox2.Text);
                        double m= j*k;
                        textBox3.Text= m.ToString();}
                  if (radioButton4.Checked){
                   double j= double.Parse(textBox1.Text);
                   double k=double.Parse(textBox2.Text);
                   try{
                    double m= j/k;
                    textBox3.Text= m.ToString();}
                    catch(System.ArithmeticException  ee)
                     {MessageBox.Show ("Διαιρεση με το μηδεν");}}}
```

10.3 ΑΣΚΗΣΕΙΣ

1. Να γραφεί παραθυρική εφαρμογή όπου ο χρήστης θα επιλέγει το χρώμα που επιθυμεί από μια λίστα και αυτόματα θα μετατρέπει το πρόγραμμα ένα label στο χρώμα αυτό.
2. Να γραφεί παραθυρική εφαρμογή όπου ο χρήστης θα κατασκευάζει το **calculator των windows.**
3. Να γραφεί παραθυρική εφαρμογή όπου ο χρήστης θα κατασκευάζει το **notepad των windows.**
4. Να γραφεί παραθυρική εφαρμογή όπου ο χρήστης θα κατασκευάζει τον **windows explorer.**

ΚΕΦΑΛΑΙΟ 11ο

11. C# ΚΑΙ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

11.1 ΕΙΣΑΓΩΓΗ

Σχετικά με τις κλασσικές βάσεις δεδομένων που βασίζονται σε πολύπλοκες ή μη δομές δεδομένων η C# στο σημείο αυτό, ακολουθεί κανονικά τη Java με random access τεχνικές και αντίστοιχες κλάσεις, θέματα στα οποία εξετάστηκαν στο ειδικό κεφάλαιο C# και αρχεία.

Στη συνέχεια, αυτό που θα μας απασχολήσει είναι αποκλειστικά το θέμα της εκτέλεσης της C# με **σχεσιακές βάσεις δεδομένων (RDBMS)** που βασίζονται σε SQL άλγεβρα, αλλά και σε άλλες μορφές δεδομένων όπως είναι τα επίπεδα αρχεία (flat files).

Η Java για σχέσεις με RDBMS χρησιμοποιεί το **API JDBC** με μια πληθώρα δυνατοτήτων. Η C# αντίστοιχα, χρησιμοποιεί το **ADO.NET** το οποίο θεωρείται όχι μόνο ισοδύναμο του JDBC αλλά υπερτερεί σε σχέση με αυτό σε παρεχόμενες ευκολίες, προσπελάσεις και ασφάλεια. Το ADO.NET συγκροτείται από δύο βασικά συστατικά. Τον **Data Provider** ο οποίος περιλαμβάνει κάθε διοίκηση στις πηγές των **data,** αλλά και προσπελάσεις αυτών. Το δεύτερο συστατικό είναι το **DataSet** το οποίο λειτουργεί αποκλειστικά σε επίπεδο μνήμης σαν μια αποθήκη δεδομένων και διαθέτει τεχνικές συσχέτισης και σύνθεσης δεδομένων από διάφορες πηγές. Επίσης, το DataSet, οποτεδήποτε χρειαστεί μπορεί να σταλεί για κανονική αποθήκευση των δεδομένων αυτών.

Αν και ορισμένοι συνδέουν το API JDBC της Java με το Data Provider, ωστόσο, υπάρχει και σχέση του DataSet με το **RowSet** ενδιάμεσο της Java. Οπωσδήποτε, αν και υπάρχον αυτές οι σχέσεις που αναφέρθηκαν, ωστόσο είναι φανερό, ότι δεν μπορεί να γίνει άμεση σύγκριση μεταξύ αυτών. Γι αυτό το λόγο είναι αναγκαίο στη συνέχεια να αναπτυχθεί το Data Provider και το DataSet αποκλειστικά χωρίς να γίνεται συσχέτιση με το RowSet και το JDBC της Java. Τέλος, οτιδήποτε αναφέρεται στο κεφάλαιο αυτό, υποτίθεται ότι ο αναγνώστης γνωρίζει τα βασικά σε σχεσιακές βάσεις δεδομένων.

11.2 Data Provider

Το Data Provider, αποτελείται από τις ακόλουθες στοιχειώδεις δραστηριότητες όπως φαίνεται στο παρακάτω πίνακα:

ADO.NET	**JDBC**
Connection	java.sql.Connection
Command	java.sql
Data Reader	ResultSet interface
Data Adapter	Επικονωνία DataSet- Data Provider

Ειδικότερα, το **Connection** παρέχει σύνδεση με το Data Source, έλεγχο **transaction** και δημιουργία **command components.**Η δραστηριότητα **Command** εμπεριέχει διαχείριση δεδομένων και εντολές **ερωταποκρίσεων (queries)**. Επίσης παρέχει υποστήριξη για διαχείριση παραμετρικών ερωταποκρίσεων. Ο **Data Reader** αντίστοιχα, παρέχει μια **μέθοδο ανάγνωσης**

των αποτελεσμάτων μιας εκτελέσιμης ερωταπόκρισης. Τέλος, ο **Data Adapter**, αποτελεί μια **γέφυρα** ανάμεσα στο Data Provider και το DataSet.

Όπως αναλύθηκε, δεν υπάρχει στενή συσχέτιση στα θέματα των RDBMS με Java και C#, ωστόσο, μπορεί να γίνει κάποιος συσχετισμός μεταξύ των πακέτων της Java, java.sql και τις διαπροσωπείες του .NET ειδικότερα του namespace της C# System.Data.Τα παραπάνω φαίνεται συνοπτικά στον ακόλουθο πίνακα

Java Interface	**.NET Interface**	**.NET Class**
Connection	IDbConnection	OleDbConnection SqlConnection
Statement, PrepareSttement, CallableStatement	IdbCommand	OleDbCommand SqlCommand
ResultSet	IDataReader	OleDbDataReader SqlDataReader

Ο παραπάνω πίνακας περιέχει κλάσεις από το **System.Data.SqlClient** και από το **System.Data.OleDb**. namespace του .NET. Αυτές οι δύο κλάσεις παρέχουν διαπροσωπείες οι οποίες θα αναλυθούν στη συνέχεια.

- **OLE DB.NET Data Provider** παρέχει σύνδεση με οποιοδήποτε Data Source που χρησιμοποιεί OLE DB Interface. Όλα τα ονόματα των κλάσεων χρησιμοποιούν μπροστά το πρόθεμα OleDb.
- **SQL Server.NET Data Provider** παρέχει βέλτιστη σύνδεση με Microsoft SQL Server 7.0 και μετέπειτα εκδόσεις. Όλα τα ονόματα των κλάσεων χρησιμοποιούν το πρόθεμα Sql.

Στη συνέχεια, ακολουθεί ένα απλό πρόγραμμα καταχώρισης δεδομένων σε μια βάση. Η βάση αυτή, περιέχει αρχικά ένα πίνακα ο οποίος περιλαμβάνει τα δεδομένα μιας φαρμακευτικής εταιρείας και ειδικότερα **κωδικό φαρμάκου (code), όνομα φαρμάκου (medname), απόθεμα φαρμάκου (stock), τιμή φαρμάκου (price) και αν είναι εισαγόμενο ή εγχώριο (medimport)**. Αρχικά, για να τρέξει αυτή η παραθυρική εφαρμογή, κατασκευάζουμε την βάση μας χρησιμοποιώντας το **MS SQL SERVER 2000.** Με τη βοήθεια αυτού του εργαλείου κατασκευάζουμε εύκολα τη βάση δεδομένων μας. Στη συνέχεια παρουσιάζεται ο κώδικας στον οποίο προσθέτουμε ένα νέο φάρμακο στη βάση, ενώ παράλληλα εμφανίζονται όλα τα φάρμακα στην οθόνη με τη βοήθεια ενός ειδικού component, του **DataGrid**.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;

namespace WindowsApplicationserver
{

public class Form1 : System.Windows.Forms.Form
{
private System.Windows.Forms.DataGrid dataGrid1;
```

```
private System.ComponentModel.Container components = null;
private SqlDataAdapter dadapt;
private DataSet ds;
private DataTable dt;
private SqlConnection con;

public Form1()
{
```

/*Αρχικά δημιουργούμε ένα string το οποίο περιέχει πληροφορίες για τη σύνδεση με το Server. Ειδικότερα αποθηκεύουμε στο string αυτό παραμέτρους σύνδεσης, όπως, το όνομα του Server, login/password administrator και, τέλος το όνομα της βάσης μας.*/

```
InitializeComponent();
string ConnString="server=AGGELIKH; uid=AGGELIKH; pwd=pavlos;
database=medicine";
```

/*Ένα δεύτερο σημείο που πρέπει να επισημάνουμε, είναι η δημιουργία ενός string που περιέχει την SQL ερώτηση. Η ερώτηση αυτή, θα χρησιμοποιηθεί αργότερα για να εμφανίσουμε τα δεδομένα της βάσης στο παράθυρο της εφαρμογής μας. Στη συνέχεια υπάρχει ένα δεύτερο string που δίνει την SQL εντολή προκειμένου να εισάγουμε ένα φάρμακο στη βάση μας.*/

```
string medselect="SELECT * FROM meditem";
string medselect1="INSERT INTO meditem
(code,medname,stock,price,medimport)VALUES
('123S','VOMEX',3,3.20,1)";
```

/*Δημιουργούμε ένα connection, το οποίο το αρικο0ποιούμε και το οποίο περιέχει το string με τα δεδομένα για σύνδεση στο Server. Στη συνέχεια, δημιουργούμε ένα αντικείμενο, τύπου command, το οποίο το αρχικοποιούμε και το οποίο περιλαμβάνει την εντολή SQL για διαχείριση της βάσης, καθώς και το connection με το οποίο συνδεόμαστε με το Server. Τέλος, ανοίγουμε τους διαύλους επικοινωνίας.*/

```
SqlConnection con=new SqlConnection(ConnString);
SqlCommand cmd=new SqlCommand(medselect,con);
con.Open();
```

/*Δημιουργούμε ένα αντικείμενο το οποίο θα περιέχει όλα τα δεδομένα που εξάγονται από τη βάση. Επιπλέον, δημιουργούμε ένα DataSet και στη συνέχεια, γεμίζουμε το DataSet αντικείμενο με τα δεδομένα που εξάγαμε από τη βάση. Τέλος, κλείνουμε τη σύνδεση. Στη συνέχεια εκτελούμε την εντολή insert με παρόμοιο ακριβώς τρόπο, όπως εκτελέσαμε την εντολή select */

```
SqlDataAdapter dadapt= new SqlDataAdapter(medselect, ConnString);
DataSet ds= new DataSet();
dadapt.Fill(ds,"meditem");
con.Close();
con.Open();
SqlCommand cmd1=new SqlCommand(medselect1,con);
SqlDataAdapter dadapt1= new SqlDataAdapter(medselect1, ConnString);
dadapt1.Fill(ds,"meditem");
```

/*Το τελευταίο σημαντικό κομμάτι στον κώδικα αυτό, είναι η δημιουργία ενός πίνακα από το DataSet και η τοποθέτηση των δεδομένων του

πίνακα στο DataSet ώστε να εμφανιστούν στο DBGrid της παραθυρικής εφαρμογής μας.*/

```
DataTable dt=ds.Tables[0];
dataGrid1.DataSource=ds.Tables["meditem"].DefaultView;
con.Close();}

protected override void Dispose( bool disposing ){
if( disposing ){
      if (components != null) {
      components.Dispose();
      }
}
base.Dispose( disposing );
}
#region Windows Form Designer generated code
private void InitializeComponent(){
this.dataGrid1 = new System.Windows.Forms.DataGrid();
((System.ComponentModel.ISupportInitialize)
(this.dataGrid1)).BeginInit();
this.SuspendLayout();
this.dataGrid1.DataMember = "";
this.dataGrid1.HeaderForeColor =
System.Drawing.SystemColors.ControlText;
this.dataGrid1.Location = new System.Drawing.Point(16, 8);
this.dataGrid1.Name = "dataGrid1";
this.dataGrid1.ReadOnly = true;
this.dataGrid1.Size = new System.Drawing.Size(504, 280);
this.dataGrid1.TabIndex = 0;

this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.ClientSize = new System.Drawing.Size(536, 326);
this.Controls.AddRange(new System.Windows.Forms.Control[]{
this.dataGrid1});
this.Name = "Form1";
this.Text = "Form1";

((System.ComponentModel.ISupportInitialize)(this.dataGrid1)).EndInit(
);
this.ResumeLayout(false);}
#endregion
[STAThread]
static void Main()
{
Application.Run(new Form1());
}}}
```

Ορισμένες χρήσιμες παράμετροι για να ορίσουμε την κλάση **SqlConnection** δίνονται στον παρακάτω πίνακα:

Παράμετροι	**Εξήγηση**
Application Name	Το όνομα της εφαρμογής που χρησιμοποιείται για σύνδεση
Connect Timeout	Ο χρόνος προσπάθειας σύνδεσης πριν αποτύχει τελείως
Data Source	Το όνομα ή η διεύθυνση του SQL Server.
Database	Το όνομα της βάσης δεδομένων

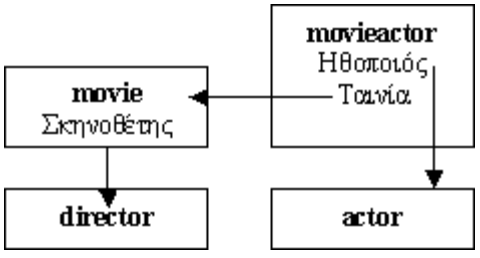
Integrated Security	Καθορίζει αν η σύνδεση είναι ασφαλής
Packet Size	Μέγεθος πακέτων δικτύου σε Bytes
Password	Passwsord για σύνδεση σε SQL Server
User ID	User ID για σύνδεση σε SQL Server
Workstation ID	Ταυτότητα σταθμού εργασίας που χρησιμοποιείται για τη σύνδεση

Ένα ιδιαίτερα χρήσιμο component για επικοινωνία με τη βάση δεδομένων είναι το DataSet. Ορισμένες από τις λειτουργίες του DataSet, φαίνονται στον παρακάτω πίνακα.

Ενέργεια	**Ερμηνεία**
DataSet.Tables.Add(TableName)	Προσθήκη πίνακα
DataSet.Tables[TableName] DataSet.Tables[TableIndex]	Απόκτηση ενός συγκεκριμένου πίνακα
DataSet.Tables.Clear()	Διαγραφή όλων των πινάκων
DataSet.Tables.Remove(TableName) DataSet.Tables.RemoveAt(TableIndex)	Διαγραφή ενός συγκεκριμένου πίνακα
DataTable.Columns.Add[ColumnName DataTable.Columns.Add(DataColumn)	Προσθήκη στήλης πίνακα
DataTable.Columns[ColumnName] DataTable.Columns[ColumnIndex]	Απόκτηση μιας συγκεκριμένης στήλης
DataTable.Columns.Clear()	Διαγραφή όλων των στηλών
DataTable.Columns.Remove(ColumnName) DataTable.Columns.RemoveAt(ColumnIndex)	Διαγραφή μιας συγκεκριμένης στήλης
DataTable.Rows.Add(DataRow) DataTable.Rows.InsertAt(DataRow, Rowindex)	Προσθήκη γραμμής πίνακα
DataTable.Rows[RowIndex]	Απόκτηση μιας συγκεκριμένης γραμμής
DataTable.Rows.Clear()	Διαγραφή όλων των γραμμών
DataTable.Rows.Remove(DataRow) DataTable.Rows.RemoveAt(RowIndex)	Διαγραφή μιας συγκεκριμένης γραμμής
DataRow[ColumnName] DataRow[ColumnIndex]	Επεξεργασία δεδομένων ανά γραμμή

11.3 ΑΣΚΗΣΕΙΣ

1. Το παρακάτω σχήμα αποτελεί τη βάση ταινιών ενός Video Club.



Movie.dbs (μέχρι 1000 εγγραφές)

Τίτλος	50 χαρακτήρες
Έτος παραγωγής	Ακέραιος

movieactor.dbs

Κωδικός Ηθοποιού	Ακέραιος
Κωδικός Ταινίας	Ακέραιος

Κωδικός Σκηνοθέτη Ακέραιος
Αριθμός Oscar Ακέραιος

actor.dbs (μέχρι 1000 εγγραφές)
Ηθοποιός 50 χαρακτήρες
Αριθμός Oscar Ακέραιος

director.dbs
Σκηνοθέτης 50 χαρακτήρες
Αριθμός Oscar Ακέραιος

Ζητείται πρόγραμμα C++ που να διαβάζει κωδικούς ηθοποιών από ASCII αρχείο codes.txt, μία τιμή σε κάθε γραμμή, και να τυπώνει στην οθόνη το όνομα του ηθοποιού, μόνο αν αυτός έχει πάρει Oscar, τα Oscar που έχει πάρει, και τις ταινίες στις οποίες συμμετείχε τα τελευταία 5 χρόνια. Η δομή της εξόδου είναι (για κάθε ηθοποιό): μία γραμμή για το όνομά του, και από κάτω μία ταινία ανά γραμμή, η οποία περιέχει τον τίτλο, το σκηνοθέτη και τον αριθμό Oscar της ταινίας.

Για τα αρχεία movie, director και actor, ο κωδικός εγγραφής ταυτίζεται με τη θέση αυτής στο αρχείο. Το αρχείο movieactor πραγματοποιεί τη σύνδεση ανάμεσα στις ταινίες και τους ηθοποιούς. Δηλαδή, για κάθε δεδομένο ηθοποιό και δεδομένη ταινία, είτε υπάρχει ένα ζεύγος των κωδικών τους στο movieactor, οπότε ο ηθοποιός παίζει στην ταινία, ή δεν υπάρχει, οπότε ο ηθοποιός δεν παίζει στην ταινία. Αυτά τα ζεύγη κωδικών δείχνουν τις αντίστοιχες εγγραφές στα αρχεία movie και actor. Αντίστοιχα, το πεδίο Σκηνοθέτης του αρχείου movie δείχνει στην αντίστοιχη εγγραφή του αρχείου director.

2.Να δημιουργηθεί εφαρμογή προσομοίωσης της λειτουργίας ενός ATM μηχανήματος τραπέζης.

ΚΕΦΑΛΑΙΟ 12ο

12. ΕΙΔΙΚΑ ΘΕΜΑΤΑ ΣΕ C#

12.1 ΕΙΣΑΓΩΓΗ

Ένα τελευταίο κομμάτι που θα αναλυθεί είναι η δυνατότητα που έχει η C# να επικοινωνεί άριστα με μεταγλώσσες προγραμματισμού όπως **η XML**, η μεγάλη και εύκολη χρήση του διαδικτυου με την **αποστολή πακέτων** ή την δημιουργία εύχρηστων δικτυακών εφαρμογών **(WEB SERVICES).**

12.2 C# ΚΑΙ XML

Η XML αποτελεί την κορυφαία μεταγλώσσα, για την μεταφορά μετάδεδομένων στο Web. Χρησιμοποιεί υψηλά στάνταρ που την κάνουν ισχυρή και μοναδική στο χώρο αυτό. Η C# ενσωματώνει πλήρως την XML, τα στάνταρ που αυτή χρησιμοποιεί και επικοινωνεί μαζί της άριστα μέσω των κλάσεων και των namespaces. Η βιβλιοθήκη που χρησιμοποιείται για την επίτευξη αυτής της σύνδεσης είναι η **System.Xml.** Μερικές από τις μεθόδους του πακέτου αυτού, φαίνονται στον παρακάτω πίνακα:

Όνομα κλάσης	**Περιγραφή**
XmlReader	Παρέχει γρήγορα XML δεδομένα
XmlWriter	Γράφει γρήγορα XML δεδομένα σε κάποιο ρεύμα
XmlTextReader	Παρέχει γρήγορη προώθηση πρόσβασης μέσω ρεύματος σε XML δεδομένα
XmlTextWriter	Γράφει γρήγορα XML δεδομένα σε ρεύμα
XmlNode	Κλάση που παρουσιάζει ένα απλό κόμβο σε ένα XML αρχείο
XmlDocument	Παρουσιάζει σε δενδροειδή μορφή το DOM που αποτελεί αναπαράσταση ενός XML αρχείου
XmlDataDocument	Έγγραφο που μπορεί να δημιουργηθεί μέσα από XML δεδομένα ή από δεδομένα ενός RDBMS με χρήση ADO.NET
XmlResolver	Αναλύει XML κείμενα όπως DTD ή επεξεργάζεται XSL αντικείμενα
XmlUrlResolver	Αναλύει εξωτερικές πηγές μέσω Url.

Παράλληλα η XML είναι κομμάτι του **System.Data** και τμήμα της κλάσης **DataSet.** Ειδικότερα έχουμε:

Όνομα κλάσης	**Περιγραφή**
ReadXml	Διαβάζει XML δεδομένα και σχήμα μέσα από ένα DataSet
ReadXmlSchema	Διαβάζει XMLSchema μέσα από DataSet

WriteXml	Γράφει XML και σχήμα από DataSet σε XML έγγραφο
WriteXmlSchema	Γράφει το σχήμα από DataSet σε XML αρχείο

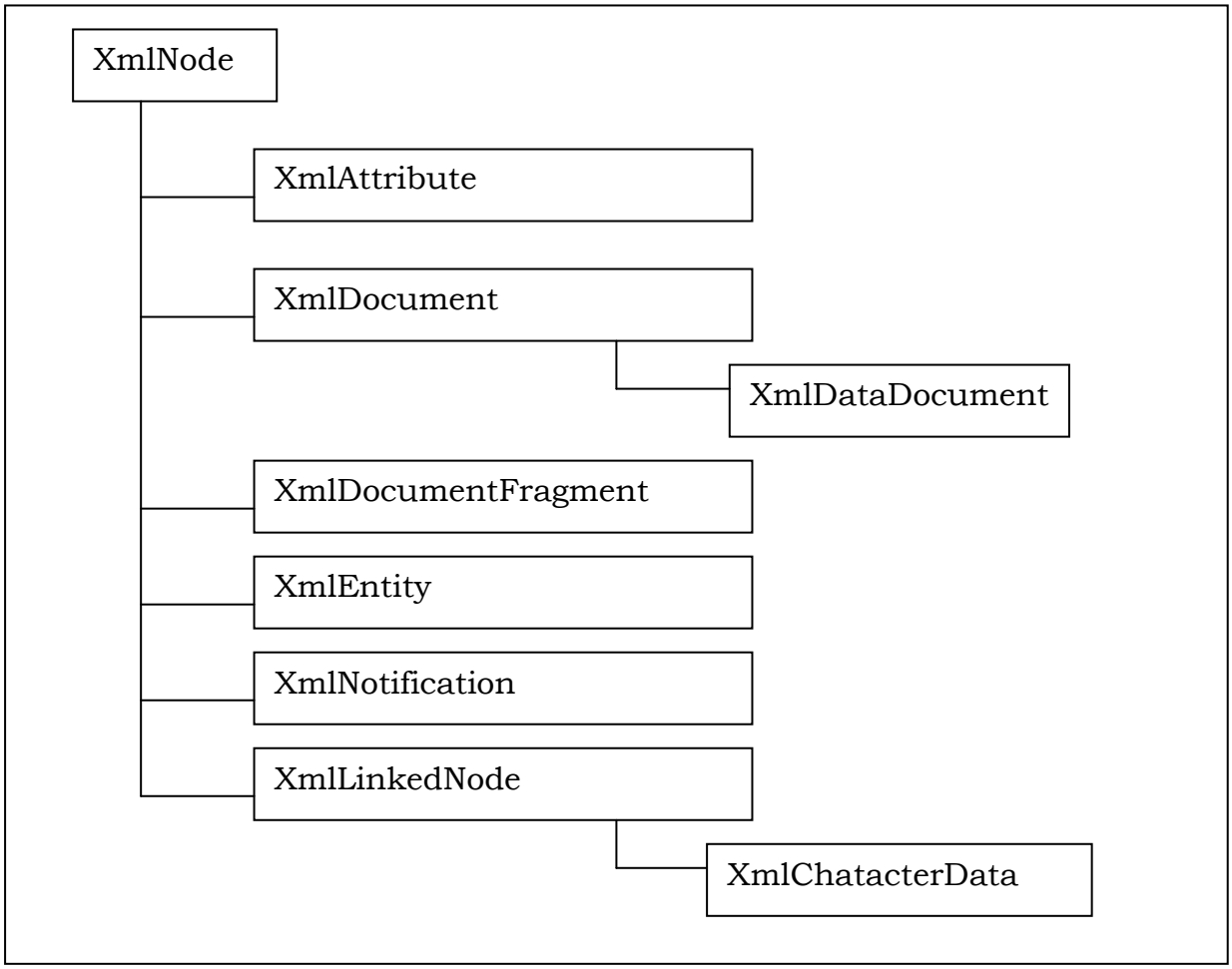
Ακολουθεί ένα παράδειγμα δημιουργίας εφαρμογής δισκοπωλείου. Αρχικά, δημιουργούμε το XML αρχείο με ονομασία disk.xml η δομή του αρχείου φαίνεται παρακάτω

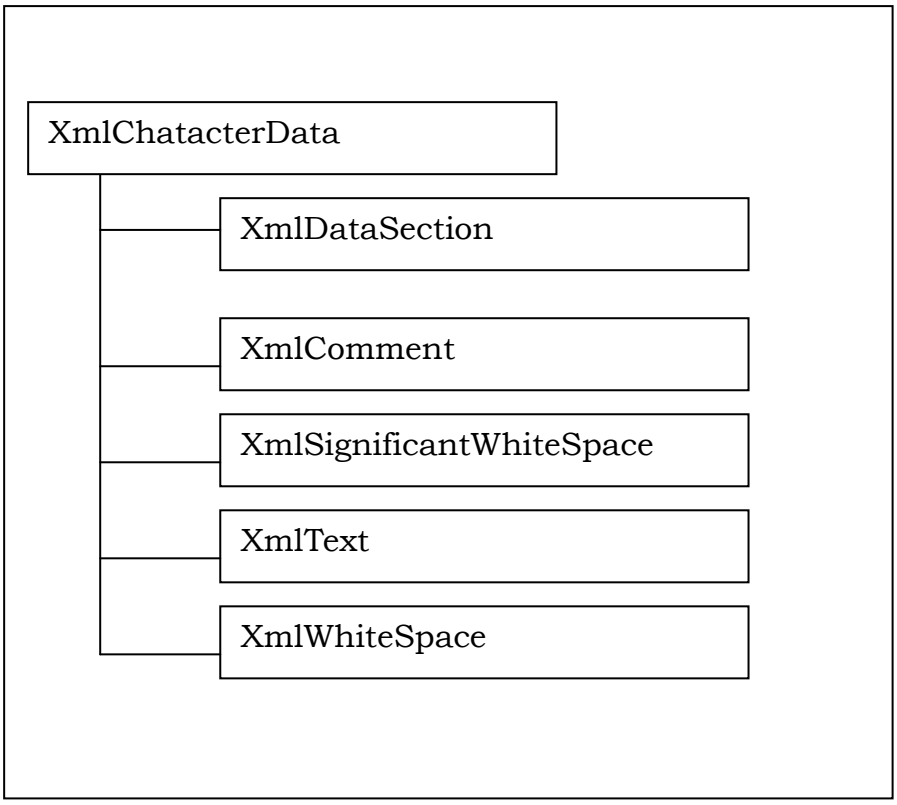
```
<?xml version= "1.0"?>
<DISKSHOP>
<DISK COVER="BLUE" LETTERS="ARIAL" SHAPE="RECTANGLE">
<TITLE> Best Of  '90s </TITLE>
<SINGER TYPE="SOLO" ARTISTS="20">  Various </SINGER>
<PRICE> 20.30€ </PRICE>
</DISK>
<DISK COVER="GREEN" LETTERS="ARIAL BOLD" SHAPE="RECTANGLE">
<TITLE> Paradise City </TITLE>
<SINGER TYPE="BAND" ARTISTS="FOUR"> Guns n' Roses </SINGER>
<PRICE> 35.82€ </PRICE>
</DISK>
</DISKSHOP>
```

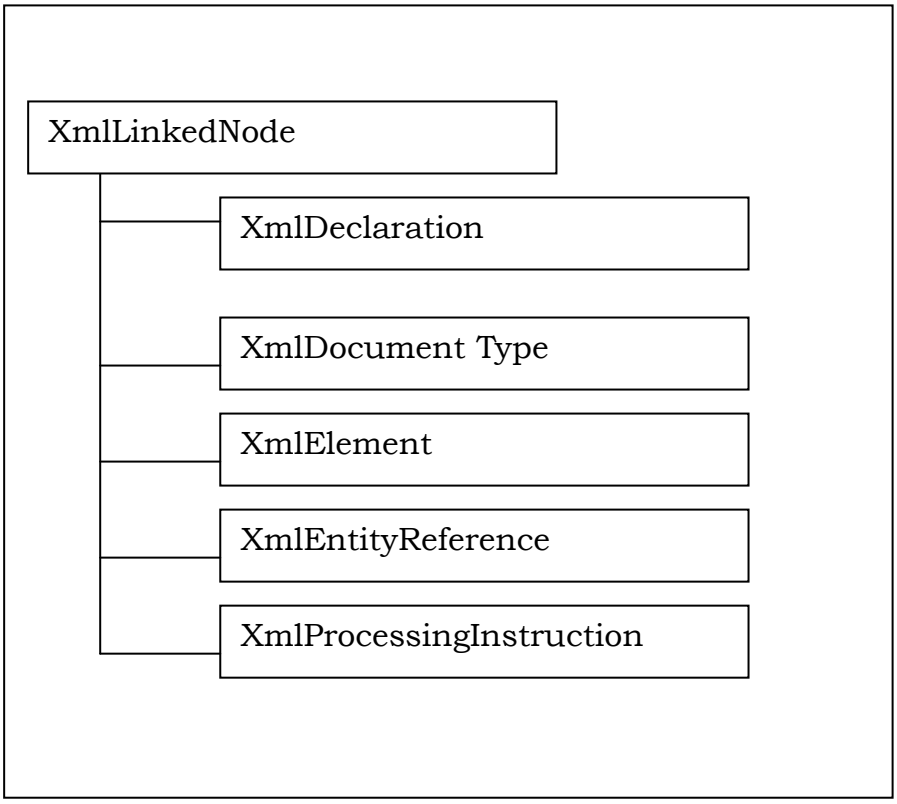
Στη συνέχεια, γράφουμε μια μικρή εφαρμογή, με την οποία μπορούμε να εμφανίζουμε τα **χαρακτηριστικά (attributes)** κάθε **στοιχείου (element)**.

```
using System;
using System.Data;
using System.Xml;

namespace ConsoleApplicationxml
{
      class Class1
      {
      [STAThread]
      static void Main(string[] args)     {
      String MyFile="C:\\xmlparadeigma\\disk.xml";
      XmlTextReader rdr=new XmlTextReader(MyFile);
      while(rdr.Read()){
      if(rdr.NodeType==XmlNodeType.Element){
      Console.WriteLine("Inspectting Node:{0}", rdr.Name);
            if (rdr["DISK"]!=null)  {
            Console.WriteLine("\tDISK={0}",rdr["DISK"]);}
            else {
while(rdr.MoveToNextAttribute()){

Console.WriteLine("\t{0}={1}",rdr.Name,rdr.Value);}
rdr.MoveToElement();
}}}}}
```

Ειδικότερα παρουσιάζουμε ένα διάγραμμα πάνω στο οποίο στηρίζεται η κλάση **XmlNode**, με βάση την οποία δομείται το **Document Object Model (DOM)** που αποτελεί αναπαράσταση των κόμβων ενός xml κειμένου.


XmlNode
XmlAttribute
XmlDocument
XmlDataDocument
XmlDocumentFragment
XmlEntity
XmlNotification
XmlLinkedNode
XmlChatacterData


XmlChatacterData
XmlDataSection
XmlComment
XmlSignificantWhiteSpace
XmlText
XmlWhiteSpace



12.3 C# ΚΑΙ ΔΙΚΤΥΑ

Σε αυτήν την παράγραφο, θα προσπαθήσουμε να δείξουμε πως δουλεύει το Microsoft .NET framework σε δικτυακές εφαρμογές και τις διαφορές που παρουσιάζει με το Java.NET πακέτο.

Υπηρεσία Δικτύου	**Java**	**.NET**
Request/Response	URLConnection	System.Net.WebClient System.Net.WebRequest System.Net.WebResponse
Protocol	Socket ServerSocket DatagramSocket MulticastSocket	System.Net.Sockets.TcpClient System.Net.Sockets.TcpListener System.Net.Sockets.UdpClient
Native Socket Access	N/A	System.Net.Sockets.Socket

Αναλυτικότερα, για τα sockets πρέπει να επισημάνουμε την δυνατότητα που έχουν να εργάζονται μαζί με τη βοήθεια των πρωτοκόλλων TCP, UDP. Για να μπορέσουμε αν καταλάβουμε τη χρήση των TCP Client κλάσεων, παραθέτουμε το παρακάτω παράδειγμα, στο οποίο θα προσπαθήσουμε να ανοίξουμε μια σύνδεση με έναν HTTP Server, θα στείλουμε ένα μήνυμα εμφάνισης μιας ηλεκτρονικής σελίδας και θα εκτυπώσουμε την απάντηση του Server. Αφού ο Server επεξεργαστεί την αίτηση και αποστείλει την απάντηση στον Client, τα ρεύματα και τα Sockets θα κλείσουν.

```
using System;
using System.Net.Sockets;
using System.IO;
```

```
namespace ConsoleApplicationSocket
{
      class ClassCSocket
      {
            ClassCSocket(){
            TcpClient cclient= new TcpClient("www.unipi.gr", 80);
            NetworkStream cstream=cclient.GetStream();
            StreamWriter cwriter= new StreamWriter(cstream);
            StreamReader creader= new StreamReader(cstream);
            cwriter.WriteLine("GET/HTTP/1.0");
            cwriter.WriteLine();
            cwriter.Flush();
            string str;
                  while((str=creader.ReadLine())!=null){
                  Console.WriteLine(str);}
                  cwriter.Close();
                  creader.Close();
                  cstream.Close();
                  cclient.Close();
            }
            [STAThread]
            static void Main(string[] args)
            {
            new ClassCSocket();
            }}}
```

12.4 C# ΚΑΙ WEB-SERVICES

Τα Web Services αποτελούν μια πρωτοπόρα διαδικασία μέσα από την οποία μπορούμε να χτίσουμε εφαρμογές, να διευρύνουμε τον ορίζοντα των κατανεμημένων εφαρμογών και γενικότερα να εκμεταλλευτούμε τα προτερήματα του Internet. Τα Web Services είναι μικρές υπηρεσίες οι οποίες καλύπτουν ανάγκες και υλοποιούν εφαρμογές μέσα από το δίκτυο.

Η μεταφορά δεδομένων μέσω Web Services γίνεται με τη χρήση πρωτοκόλλων του Internet ειδικά καθορισμένων για εύρεση, έκθεση και επεξεργασία Web εφαρμογών. Τέτοια πρωτόκολλα είναι το **SOAP (Simple Object Access Protocol)** το οποίο είναι ένα απλό πρωτόκολλο μεταφοράς μηνυμάτων χτισμένο πάνω σε **XML, HTTP, SMTP.** Χρησιμοποιεί όλα τα προνομία που του παρέχει η XML και τα στάνταρ πρωτοκόλλων επικοινωνίας του Internet.

Δύο άλλα πρωτόκολλα για μεταφορά δεδομένων είναι το **WSDL (Web Service Description Language)** το οποίο αναπτύχθηκε από τη **Microsoft** και την **IBM** και το οποίο στηρίζεται σε ένα xml σχήμα το οποίο περιγράφει τις μεθόδους ενός Web Service. Το δεύτερο πρωτόκολλο είναι το **UDDI** το οποίο και αυτό παρέχεται από την **Microsoft** και την **IBM** με χρήση παρόμοια με το **WSDL.** Η μέθοδος **Discovery** που περιέχει είναι χρήσιμη για την εύρεση μέσω του URL πληροφοριών, τοποθεσίας και άλλων στοιχείων ενός απομακρυσμένου Web Service.

Για να κατασκευάσουμε ένα Web Service, αλλά και για να μπορέσουμε αργότερα να το εντοπίσουμε μέσα στο δίκτυο το .NET μας παρέχει την κλάση **System.Web.Services.WebService.** για να δημιουργήσουμε τον κατάλληλο κώδικα ενός Web Service, δεν χρειάζεται παρά να προσθέσουμε στο σημείο του κώδικα τη δεσμευμένη λέξη **[WebMethod]**. Ένα απλό παράδειγμα για να

μπορέσουμε να κατανοήσουμε τη χρήση των Web Services είναι το παρακάτω στο οποίο ο χρήστης δίνει δύο νούμερα και υπολογίζονται το άθροισμα η διαφορά και το γινόμενο τους.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace ConsoleApplication15
{[WebService(OnomaServerService)]
      class Class1: System.Web.Services.WebService
            {
       public Class1(){
       InitializeComponent();
       }
       #region Component Designer generated code
       private void InitializeComponent(){}
       #endregion
       public override void Dispose(){}
       [WebMethod]
       public double add(double x, double y){
       return x+y;}
       [WebMethod]
       public double sub(double x, double y){
             return x-y; }
       [WebMethod]
       public double multi(double x, double y){
             return x*y;}}}
```

12.5 ΑΣΚΗΣΕΙΣ

1. Να αναπτυχθεί πλήρως ένα ηλεκτρονικό βιβλιοπωλείο.
2. Να δημιουργηθεί ένα web service ενός scientific calculator.
3. Να γραφεί πρόγραμμα το οποίο να παρέχει μετατροπή από Euro, σε δολάρια. Το παράθυρο της οθόνης να διαθέτει τρία πλαίσια δεκαδικών αριθμών και ένα πλήκτρο επιβεβαίωσης (OK).Ο χρήστης θα πρέπει να εισάγει μια μόνο δεκαδική τιμή σε κάποιο πλαίσιο και στην συνέχεια αφού δώσει την επιβεβαίωση να εμφανίζονται οι αντίστοιχες τιμές. Η ισοτιμία δολαρίου Euro είναι 1.1 δολάρια για κάθε Euro.