ΤΕΙ ΠΑΤΡΑΣ

ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ

Τμήμα Επιχειρηματικού Σχεδιασμού και Πληροφοριακών Συστημάτων

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μελέτη του περιβάλλοντος προγραμματισμού .NET

Εισηγητής: Στάμος Κωνσταντίνος

Σπουδαστές: Παπακωνσταντόπουλος Νικόλαος

Χριστιάς Αλέξανδρος

Περιεχόμενα

Πρόλογος

Η εργασία με θέμα «Μελέτη του περιβάλλοντος προγραμματισμού .NET» έχει ως σκοπό να παρουσιάσει την αρχιτεκτονική του .NET Framework, την σύγκριση της πλατφόρμας του .NET με τις πλατφόρμες της Java καθώς και να εξοικειώσει τους αναγνώστες με την έννοια της γλώσσας προγραμματισμού C#, το πώς αυτή λειτουργεί και ποια τα συστατικά της.

Η εργασία εκπονήθηκε το πρώτο εξάμηνο του 2010. Η εκπόνηση της εργασίας πραγματοποιήθηκε από τους σπουδαστές Παπακωνσταντόπουλο Νικόλαο και Χριστιά Αλέξανδρο, με επιβλέποντα τον καθηγητή κ. Κωνσταντίνο Στάμο.

Ιδιαίτερες ευχαριστίες στον εισηγητή της πτυχιακής μας κ. Κωνσταντίνο Στάμο για την καθοδήγηση και την παρότρυνση που μας παρείχε κατά τη διάρκεια εκπόνησης της πτυχιακής εργασίας.

Πάτρα 2010

Εισαγωγή

Το Microsoft .NET είναι ένα πλαίσιο λογισμικού που μπορεί να εγκατασταθεί σε υπολογιστές που τρέχουν λειτουργικά συστήματα των Microsoft Windows. Περιλαμβάνει μια τεράστια βιβλιοθήκη κωδικοποιημένων λύσεων σε κοινά προγραμματιστικά προβλήματα και μία εικονική μηχανή η οποία χειρίζεται την εκτέλεση των προγραμμάτων που είναι γραμμένα ειδικά για αυτό το πλαίσιο. Το σύστημα .NET είναι μια προσφορά της Microsoft και πρόκειται να χρησιμοποιηθεί από τις περισσότερες εφαρμογές που έχουν δημιουργηθεί για την πλατφόρμα των Windows.

Η Πρότυπη Βιβλιοθήκη Κλάσης του συστήματος παρέχει ένα μεγάλο εύρος χαρακτηριστικών γνωρισμάτων συμπεριλαμβανομένων της αλληλεπίδρασης χρηστών, βάσης δεδομένων, συνδεσιμότητας βάσεων δεδομένων, κρυπτογραφίας, ανάπτυξης εφαρμογών ιστοσελίδων, αριθμητικούς αλγόριθμους και επικοινωνίας δικτύων. Η Βιβλιοθήκη Κλάσης χρησιμοποιείται από προγραμματιστές, που τη συνδυάζουν με ένα δικό τους κώδικα ώστε να παράγουν εφαρμογές.

Προγράμματα που είναι γραμμένα για το .NET Framework εκτελούνται σε ένα περιβάλλον λογισμικού που χειρίζεται τις απαιτήσεις χρόνου εκτέλεσης του προγράμματος . Επίσης μέρος του προγράμματος εκτέλεσης είναι το γνωστό ως Common Language Runtime(CRL). Το CRL παρέχει την εμφάνιση μια εικονικής μηχανής, έτσι ώστε οι προγραμματιστές να μη χρειάζεται να εξετάσουν τις δυνατότητες του συγκεκριμένου επεξεργαστή που θα εκτελέσει το πρόγραμμα. Το CRL παρέχει επίσης άλλες σημαντικές υπηρεσίες όπως η ασφάλεια, η διαχείριση μνήμης, και ο χειρισμός εξαίρεσης. Η Βιβλιοθήκη Κλάσης και το CRL μαζί αποτελούν το .NET Framework.

Η έκδοση 3.0 του .NET Framework περιλαμβάνεται στα Windows Server 2008, Windows Vista και Windows 7. Η τρέχουσα σταθερή έκδοση του framework, το οποίο είναι 3.5, μπορεί επίσης να εγκατασταθεί σε Windows XP και στην οικογένεια

των λειτουργικών συστημάτων Windows Server 2003. Η έκδοση 4 κυκλοφόρησε δημόσια σε δοκιμαστικό στις 20 Μαΐου του 2009.

Η οικογένεια .NET Framework περιλαμβάνει επίσης δύο εκδόσεις για τη χρήση κινητών ή ενσωματωμένων συσκευών. Μια μικρότερη έκδοση του framework, το .NET Compact Framework, είναι διαθέσιμη για πλατφόρμες Windows CE, συμπεριλαμβανομένων και των κινητών τηλεφώνων και συσκευές όπως τα smartphones. Επιπλέον, το framework .NET Microsoft, απευθύνεται αυστηρά σε πόρους περιορισμένων συσκευών.

Το .NET παρέχει μια API (Application Programming Interface), νέα λειτουργικότητα, και νέα εργαλεία για γράψιμο των Windows και για εφαρμογές Web, συστατικά και υπηρεσίες στην εποχή του Web.

Οι API στα Windows, οι βιβλιοθήκες των λειτουργιών χρησιμοποιούνταν για να γράφουν τις εφαρμογές των Windows, οι οποίες ήταν αρχικά γραμμένες σε γλώσσα C και έχουν σταδιακά αυξηθεί με την πάροδο των ετών. Κατ' αρχάς έχει αυξηθεί πάρα πολύ ο όγκος τους και δεν έχει καμία συνοχή η εσωτερική τους οργάνωση, κάτι που τις κάνει πολύ δύσχρηστες. Κατά τη διάρκεια της αύξησης του όγκου, προστέθηκαν χαρακτηριστικά κομματιαστά, συνεπώς δεν παρουσιάζει πάντα μια ενοποιημένη διεπαφή στους υπευθύνους για την ανάπτυξη, και περιέχει πολλές κληρονομικές λειτουργίες και τύπους στοιχείων(και είναι πλέον σαφώς ξεπερασμένα). Έπειτα, ένα πιο σοβαρό πρόβλημα είναι ότι οι API στα Windows ήταν αρχικά σχεδιασμένα για χρήση από προγραμματιστές της γλώσσας C. Αυτό σημαίνει ότι είναι δύσκολο να χρησιμοποιηθεί σε γλώσσες άλλες από τη C, κάτι που δεν ταιριάζει και πολύ με τις μοντέρνες αντικειμενοστραφείς μεθόδους και γλώσσες.

Το .NET, περιλαμβάνει ένα νέο αντικειμενοστραφές API σαν ένα σύνολο από κλάσεις, το οποίο θα είναι προσιτό από οποιαδήποτε γλώσσα προγραμματισμού. Θα ακολουθήσει πιο εκτενής αναφορά για το πώς μπορεί να χρησιμοποιηθεί τοFramework ώστε να γραφτούν εφαρμογές των Windows, όπως και περιγραφή των τάξεων του.

Η Microsoft πήρε κάποιες ριζικές αποφάσεις στο σχεδιασμό του .NET και έχει ενσωματώσει πολλά νέα μοναδικά χαρακτηριστικά τα οποία θα κάνουν τις εφαρμογές

γραψίματος – και ιδιαίτερα τις διανεμημένες εφαρμογές – ευκολότερες από ό,τι πρωτύτερα . Πολλές από τις νέες τεχνολογίες κρύβονται κάτω από την επιφάνεια και μπορεί να μην είναι ορατές από τον μη εξοικειωμένο χρήστη, αλλά η υποδομή των εφαρμογών των Windows και οι τεχνολογίες πάνω στις οποίες έχουν κατασκευασθεί και επικοινωνούν είναι πολύ διαφορετικές στον κόσμο του .NET.

Υπάρχει ένα ολόκληρο νέο σετ εργαλείων που επίσης εισάγεται με τη νέα έκδοση του Visual Studio, το οποίο είναι γνωστό ως Visual Studio .NET. Το Διαλογικό Περιβάλλον Ανάπτυξης (IDE) είναι εντελώς καινούριο, και η Microsoft έχει ριζικά ξεπεράσει τη Visual C++ και τη Visual Basic καθώς εισήγαγε μια εξ ολοκλήρου νέα μορφή γλώσσας τη C#. Ένα νέο μοντέλο για την κατασκευή διανεμημένων εφαρμογών χρησιμοποιώντας το Web και την XML(Extensive Markup Language) σημαίνει ότι ένα ολόκληρο πλήθος από νέα εργαλεία και τεχνολογίες απαιτούνται και είναι όλα ενσωματωμένα στο Visual Studio .NET.

Τεχνολογίες για τη δημιουργία διανεμημένων συστημάτων

Η Microsoft είναι πεπεισμένη ότι το μέλλον των Windows βρίσκεται στις διανεμημένες εφαρμογές, όπου τα διάφορα συστατικά στοιχεία μπορεί να μην βρίσκονται στις μηχανές των Windows που συνδέονται με ένα δίκτυο επιχείρησης. Με το πέρασμα των ετών η Microsoft έχει εισαγάγει νέες τεχνολογίες που στοχεύουν στη δημιουργία διανεμημένων συστημάτων, αλλά κάθε ένα από αυτά έχει τα μειονεκτήματα του σε μια ή σε άλλη περιοχή. Συνοπτικά αναφέρονται 3 από αυτές τις τεχνολογίες:Component Object Model(COM), Active Server Page(ASP) και Visual Basic.

Component Object Model

Επί σειρά ετών, η COM ήταν το πρότυπο της Microsoft για τον προγραμματισμό τμημάτων σε μια ποικιλία γλωσσών, η οποία μπορεί να κατασκευαστεί επάνω στις διανεμημένες εφαρμογές. Η COM ήταν πολύ επιτυχημένη, αλλά αντιμετώπιζε κάποια προβλήματα. Πρώτον, είναι πολύ δύσκολο να γίνει κανείς εξειδικευμένος προγραμματιστής COM, όπως και το να δημιουργήσει πολύπλοκες εφαρμογές COM είναι πολύ δύσκολο. Χρειάζεται λεπτομερής γνώση της γλώσσας C++ καθώς και των

εσωτερικών της COM και των Windows για να είναι κανείς επιτυχής. Δεύτερον, η COM είναι μια ειδική αρχιτεκτονική της Microsoft και είναι διαθέσιμη μόνο για ορισμένο αριθμό πλατφόρμων. Τρίτον, εξ αιτίας των ιδιόκτητων πρωτοκόλλων που χρησιμοποιούνται για να επικοινωνούν τα δίκτυα, με το μόνο που μπορεί να γίνει η επικοινωνία, είναι με άλλα συστατικά της COM. Αν και η ιδέα γύρω από την COM βρίσκει εφαρμογή, η υλοποίηση της περιορίζει τη διαδεδομένη ανάπτυξη των διανεμημένων συστημάτων.

Active Server Page

Η Microsoft εισήγαγε την ASP ως έναν τρόπο για τους εξυπηρετητές Web να παραδίδεται το προσαρμοσμένο περιεχόμενο με την εκτέλεση του κώδικα σεναριογραφιών που ενσωματώθηκε στην HTML μιας ιστοσελίδας. Η ASP έχει γίνει πολύ δημοφιλής, αλλά έχει ένα μειονέκτημα: υποστηρίζει μονάχα τις interpreted γλώσσες, όπως η VBScript και JScript. Αυτό έχει επιπτώσεις στην αποδοτικότητα – επειδή οι interpreted γλώσσες ερμηνεύονται στο χρόνο εκτέλεσης, και συνεπώς δεν είναι τόσο αποδοτικές όσο οι γλώσσες σύνταξης - και επίσης επειδή δεν μπορούν να χρησιμοποιηθούν άλλες γλώσσες. Εάν έχουμε ένα C++ κώδικα και θέλουμε να τον χρησιμοποιήσουμε σε μια ASP σελίδα, δεν μπορούμε επειδή η C++ δεν είναι γλώσσα interpreted.

Visual Basic

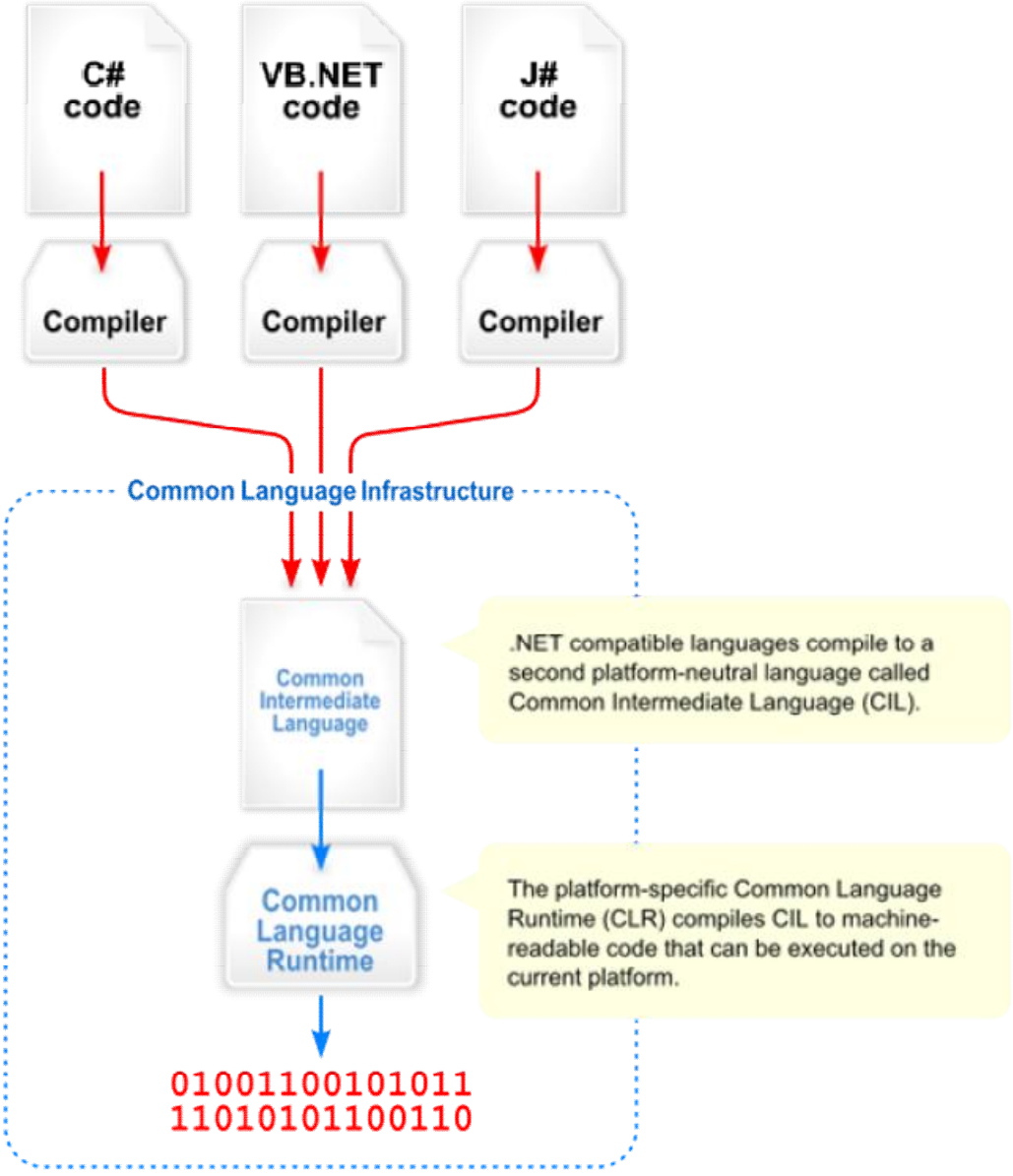
Η Visual Basic έχει απολαύσει τεράστια επιτυχία, ως η σημαντικότερη γλώσσα προγραμματισμού των Windows, με την τεχνική του συρσίματος και πτώσης (drag and drop). Ένας από τους σημαντικότερους περιορισμούς της είναι ότι είναι συνδεδεμένη με τα Windows, και έτσι δεν είναι χρήσιμη για συστήματα γραφής που είναι διανεμημένα σε ένα εύρος αρχιτεκτονικών. Επιπρόσθετα, έχει μικρή ευρύτητα δυνατοτήτων, επιτρέποντας μόνο περιορισμένη πρόσβαση πιο βαθιά στο λειτουργικό σύστημα, και δεν είναι αντικειμενοστραφής, κάτι που είναι περιορισμός για να δημιουργηθούν συστήματα μεγάλης κλίμακας. Πολλοί προγραμματιστές θα προτιμούσαν η τεχνική του συρσίματος και πτώσης(drag and drop) να ήταν διαθέσιμο και με άλλες γλώσσες προγραμματισμού των Windows, όπως η Visual C++.

Με την άφιξη του .NET, η Microsoft έχει εισαγάγει πολλές νέες τεχνολογίες οι οποίες καθιστούν το γράψιμο των βασισμένων στα συστατικά διανεμόμενων συστημάτων πιο εύκολο, πιο ευπροσάρμοστο και πιο δυνατό από ό,τι πριν. Είναι τώρα πιο εύκολο από ποτέ να γραφούν τα τμήματα σε οποιαδήποτε γλώσσα προγραμματισμού, η όποια μπορεί να επικοινωνήσει με τμήματα σε άλλες μηχανές, οι οποίες μπορεί να μη βασίζονται καθόλου στα Windows.

Κεφάλαιο 1

Αρχιτεκτονική .NET

1.1 The Common Language Infrastructure (CLI)



Οπτική αναπαράσταση της Κοινής Γλώσσας Υποδομής (CLI)

Το CLI είναι ένα πρότυπο το οποίο καθορίζει τις προδιαγραφές για τα ακόλουθα στοιχεία:

- Εικονικό Σύστημα Εκτέλεσης (VES)
- Κοινή Ενδιάμεση Γλώσσα (CIL)
- Κοινό Σύστημα Τύπων (CTS)
- Κοινή Προδιαγραφή Γλώσσας (CLS)
- Framework

Η Κοινή Γλώσσα Υποδομής (CLI) είναι μια ανοιχτή προδιαγραφή που αναπτύχθηκε από τη Microsoft και περιγράφει τον εκτελέσιμο κώδικα και το runtime περιβάλλον που αποτελούν τον πυρήνα της Microsoft .NET Framework και την ελεύθερη και ανοικτή πηγή υλοποίησης Mono και Portable .NET. Η προδιαγραφή CLI ορίζει ένα περιβάλλον που επιτρέπει σε πολλαπλές γλώσσες προγραμματισμού υψηλού επιπέδου να χρησιμοποιηθούν σε διαφορετικές πλατφόρμες Η/Υ χωρίς να χρειάζεται να ξαναγραφούν για συγκεκριμένες αρχιτεκτονικές.

Παρακάτω θα εξετάσουμε κάθε ένα από αυτά τα στοιχεία της CLI, προκειμένου να καταλάβουμε πώς το περιβάλλον της CLI είναι αλληλένδετο.

1.1.1 Κοινή Ενδιάμεση Γλώσσα (CIL)

Σε αντίθεση με τους μεταγλωττιστές της C και της C++ οι οποίοι μεταγλωττίζουν από τον πηγαίο κώδικα σε κώδικα μηχανής μέσω μικροεπεξεργαστή, ο μεταγλωττιστής της C#, μεταγλωττίζει σε μια ενδιάμεση μορφή κώδικα byte γνωστή και ως Κοινή Ενδιάμεση Γλώσσα (Common Intermediate Language).

Αυτός ο κώδικας μπορεί ,θεωρητικά, να ληφθεί από οποιοδήποτε σύστημα όπου υπάρχει ένα CLI συμμορφούμενο με το Εικονικό Σύστημα Εκτέλεσης (VES) και να εκτελεστεί.

Επομένως δεν υπάρχει ανάγκη για μεταγλώττιση μιας εφαρμογής για καθεμία πλατφόρμα.

Η λέξη Κοινή στην Κοινή Ενδιάμεση Γλώσσα χρησιμοποιείται επειδή η συγκεκριμένη μορφή κώδικα είναι κοινή και σε άλλες γλώσσες προγραμματισμού εκτός από την C#.

Στην πραγματικότητα κάθε γλώσσα προγραμματισμού μπορεί να επιλέξει την CIL επιτρέποντας σε βιβλιοθήκες και ενότητες κώδικα από άλλες γλώσσες να εκτελεστούν από κοινού στην ίδια εφαρμογή.

Μερικές γλώσσες προγραμματισμού στις οποίες μπορεί να χρησιμοποιηθεί η CIL είναι οι Visual Basic, COBOL, SmallTalk και C++.

1.1.2 Εικονικό Σύστημα Εκτέλεσης (VES)

Η VES (συνήθως αναφέρεται ως runtime) είναι το περιβάλλον στο οποίο εκτελείται ο κώδικας byte CIL. Η VES διαβάζει τον byte κώδικα που δημιουργείται από το μεταγλωττιστή της C# και χρησιμοποιεί κάτι που ονομάζεται μεταγλωττιστής «Just In Time (JIT)» για να μεταγλωττίσει τον byte κώδικα στην μητρική μηχανή κώδικα του επεξεργαστή στον οποίο λειτουργεί.

Παράλληλα με το κώδικα, εκτελείται το μέσο runtime το οποίο ουσιαστικά διαχειρίζεται την διαδικασία εκτέλεσης.

Αυτή η διαδικασία εκτέλεσης κώδικα είναι γνωστή ως «διαχειριζόμενος κώδικας» και χειρίζεται θέματα όπως είναι η συλλογή απορριμμάτων [για να χειριστεί την κατανομή μνήμης και το λεγόμενο deallocation (απελευθέρωση πακέτου πληροφοριών ώστε να χρησιμοποιηθούν από διαφορετικό πρόγραμμα)], τη

πρόσβαση μνήμης και διάφορους τύπους ασφαλείας ώστε να διασφαλιστεί ότι ο κώδικας δε θα κάνει κάποιο λάθος.

Ένας όρος που χρησιμοποιείται συχνά σε συνδυασμό με την VES είναι η Common Language Runtime (CLR).

Αξίζει να σημειωθεί ότι η διαδικασία JIT μπορεί να προκαλέσει μια καθυστέρηση στη έναρξη μιας εφαρμογής. Μια διαθέσιμη επιλογή για να αποφευχθεί το συγκεκριμένο πρόβλημα είναι να προ-μεταγλωττιστεί ο CLI κώδικας byte στη μητρική μηχανή κώδικα χρησιμοποιώντας το NGEN (Native Image Generator), που είναι ένα εργαλείο που υλοποιεί την ahead-of-time μεταγλώττιση. Μεταγλωττίζει δηλαδή μια ενδιάμεση γλώσσα (π.χ.CIL) σε ένα εξαρτώμενο σύστημα δυαδικών.

1.1.3 Κοινό Σύστημα Τύπων (CTS) & Κοινή Προδιαγραφή Γλώσσας (CLS)

Όπως αναφέρθηκε και προηγουμένως, πολλές γλώσσες προγραμματισμού επιλέγουν την CLI επιτρέποντας για παράδειγμα κώδικα από την C# να αλληλεπιδράσει με κώδικα από την Visual Basic. Προκειμένου να επιτευχθεί η συγκεκριμένη διαδικασία, κάθε γλώσσα πρέπει να έχει το ίδιο concept για τη μέθοδο με την οποία οι τύποι δεδομένων αποθηκεύονται στη μνήμη. Η CTS ως εκ τούτου, καθορίζει το πώς μια συμβατή γλώσσα CLI θα εμφανίσει κομμάτια προτύπων αξιών των σχεδίων και τη συμπεριφορά των αντικειμένων για να εξασφαλίσει την διαλειτουργικότητα.

Η CLS είναι ουσιαστικά ένα υποσύνολο του CTS που εστιάζει στη δημιουργία διαλειτουργικών βιβλιοθηκών.

1.1.4 The Framework (Πρότυπη Κλάση και Framework Βιβλιοθηκών Κλάσεως)

Το CLI καθορίζει ένα σύνολο βάσης κλάσεων που πρέπει να είναι διαθέσιμες να εκτελέσουν τον CLI κώδικα, γνωστές και ως Base Class Library (BCL). Η BCL περιλαμβάνει API που επιτρέπουν την εκτέλεση του CIL κώδικα να αλληλεπιδράσει με το runtime περιβάλλον και το underlying λειτουργικό σύστημα.

Εκτός των άλλων υπάρχει και η Βιβλιοθήκη Κλάσης Framework. Πρόκειται για μια βιβλιοθήκη της Microsoft η οποία περιέχει APIs για την δημιουργία γραφικού περιβάλλοντος χρήστη (graphical user interfaces), εφαρμογές βάσεων δεδομένων, πρόσβαση στο διαδίκτυο και πολλά άλλα.

1.1.5 Εφαρμογές για το CLI που δεν ανήκουν στην Microsoft

Η εφαρμογή της Microsoft για το CLI καλείται .NET. το .NET ωστόσο δεν είναι η μόνη διαθέσιμη εφαρμογή. Ακόμα μια που προέρχεται από την Microsoft ονομάζεται Rotor. Προορίζεται για Windows , Mac OS και FreeBSD και είναι διαθέσιμη σε μορφή κώδικα. Το Rotor ωστόσο αποτελεί κυρίως εργαλείο μάθησης και ως τέτοιο έχει αδειοδοτηθεί σύμφωνα με τους όρους που απαγορεύουν τη χρήση του ως βάση για εμπορικές εφαρμογές. Άλλες σημαντικές εφαρμογές ανοικτού κώδικα είναι το Mono και το Dot GNU, τα οποία απευθύνονται σε Windows, Linux και Unix πλατφόρμες.

1.2 .NET assembly

Στο Microsoft .NET Framework, μια συνδεσμολογία assembly είναι μια μερικώς καταρτισμένη βιβλιοθήκη κώδικα για χρήση σε ανάπτυξη, versioning και ασφάλειας.

Υπάρχουν δυο είδη assemblies: Process assemblies (EXE) και Library assemblies (DLL).

Η process assembly αντιπροσωπεύει μια διαδικασία που θα χρησιμοποιήσει κατηγορίες που καθορίζονται στις library assemblies.

Ένα assembly μπορεί να αποτελείται από ένα η περισσότερα αρχεία. Τα αρχεία κώδικα ονομάζονται modules. Ένα assembly μπορεί να περιέχει περισσότερους από έναν κώδικα module και εφ' όσον έχει την δυνατότητα να χρησιμοποιεί διαφορετικές γλώσσες για να δημιουργήσει κώδικες modules είναι τεχνικώς δυνατόν να χρησιμοποιηθούν διαφορετικές γλώσσες για να δημιουργήσουν ένα assembly. Το Visual Studio ωστόσο δεν υποστηρίζει τη χρήση διαφορετικών γλωσσών σε ένα assembly.

1.2.1 Assembly names

Το όνομα ενός assembly αποτελείται από 4 μέρη:

1. Το σύντομο όνομα. Στα Windows αυτό είναι το όνομα του φορητού εκτελέσιμου αρχείου(Portable Executable file) χωρίς την επέκταση.
2. Τη κουλτούρα. Αυτό είναι ένα RCF 1766 αναγνωριστικό για τη τοποθεσία του assembly. Σε γενικές γραμμές, τα library και process assemblies πρέπει να είναι culture neutral. Culture πρέπει να χρησιμοποιείται μόνο για assemblies δορυφόρου.
3. Η έκδοση. Αυτή είναι μια διακεκομμένη σειρά τεσσάρων αξιών – μείζονος, ελάσσονος, κατασκευής και αναθεώρησης.
4. Ένα δημόσιο κλειδί token. Αυτό είναι ένα 64-bit hash του δημόσιου κλειδιού που αντιστοιχεί στο ιδιωτικό κλειδί που χρησιμοποιείται για την

υπογραφή του assembly. Ένα υπογεγραμμένο assembly συνηθίζεται να έχει ένα ισχυρό όνομα.

Το δημόσιο κλειδί token χρησιμοποιείται για να κάνει το όνομα του assembly μοναδικό. Με αυτό τον τρόπο, δυο assemblies με ισχυρό όνομα μπορούν να έχουν το ίδιο PE(Portable Executable) όνομα αρχείου αλλά το .NET να τους αναγνωρίζει ως ξεχωριστά assemblies.

Το σύστημα αρχείων των Windows (FAT32 και NTFS) αναγνωρίζει μόνο το όνομα αρχείου επομένως 2 assemblies με το ίδιο PE όνομα αρχείου (αλλά διαφορετική κουλτούρα, έκδοση και δημόσιο κλειδί token) δεν μπορούν να συνυπάρξουν στον ίδιο φάκελο των Windows. . Το PE είναι μια δομή δεδομένων που περιγράφει πως τα διάφορα μέρη ενός εκτελέσιμου αρχείου των Win32 ή Win64 παραμένουν ενωμένα. Επιτρέπει στο λειτουργικό σύστημα να φορτώσει το εκτελέσιμο αρχείο και να εντοπίσει τις δυναμικά συνδεδεμένες βιβλιοθήκες που απαιτούνται για να τρέξει το εκτελέσιμο αρχείο και να πλοηγηθεί ο κώδικας, τα δεδομένα και τα τμήματα πόρων που είναι μεταγλωττισμένα μέσα στο εκτελέσιμο αρχείο. Για την επίλυση αυτού του ζητήματος το .NET χρησιμοποιεί το GAC (Global Assembly Cache) το οποίο αντιμετωπίζεται ως μεμονωμένος φάκελος από το .NET CLR, αλλά στην πραγματικότητα εφαρμόζεται χρησιμοποιώντας ένθετους NTFS (η FAT32) φακέλους.

Για να αποτρέψει κακόβουλες επιθέσεις όταν ένας cracker προσπαθήσει να εισέλθει σε ένα assembly ως κάτι άλλο, το assembly είναι υπογεγραμμένο με ένα ιδιωτικό κλειδί. Ο προγραμματιστής του συγκεκριμένου assembly κρατά μυστικό το ιδιωτικό κλειδί ώστε ο cracker να μην έχει πρόσβαση σε αυτό ούτε να προσπαθεί απλώς να το μαντέψει. Έτσι, ο cracker δεν μπορεί να μετατρέψει ένα assembly σε κάτι διαφορετικό καθώς δεν έχει την δυνατότητα να υπογράψει σωστά το assembly αμέσως μετά την αλλαγή.

Η υπογραφή του assembly περιέχει τη λήψη ενός hash σημαντικών τμημάτων του assembly και στη συνέχεια την κρυπτογράφηση του hash με το ιδιωτικό κλειδί. Το υπογεγραμμένο hash αποθηκεύεται στο assembly μαζί με το δημόσιο κλειδί.

Το δημόσιο κλειδί θα αποκρυπτογραφήσει το υπογεγραμμένο hash. Όταν το CLR φορτώσει ένα ισχυρά ονομασμένο assembly, θα δημιουργήσει ένα hash από το assembly κι έπειτα θα το συγκρίνει με το αποκρυπτογραφημένο hash. Εάν η σύγκριση είναι πετυχημένη αυτό σημαίνει ότι το δημόσιο κλειδί στο αρχείο (και κατά συνέπεια το δημόσιο κλειδί token) σχετίζεται με το ιδιωτικό κλειδί που χρησιμοποιείται για να υπογράψει το assembly. Αυτό σημαίνει ότι το δημόσιο κλειδί στο assembly, είναι το δημόσιο κλειδί του εκδότη του assembly και ως εκ τούτου η πιθανότητα μιας κακόβουλης επίθεσης εκμηδενίζεται.

1.2.2 Assemblies versions

Τα .NET assemblies μπορούν να έχουν πληροφορίες της έκδοσης, που τους επιτρέπει να αποφύγουν τις περισσότερες συγκρούσεις μεταξύ των εφαρμογών που προκαλούνται από τα κοινά assemblies. Ωστόσο, αυτό δεν εξαλείφει όλες τις πιθανές συγκρούσεις μεταξύ των εκδόσεων των assemblies.

1.2.3 Assemblies and .NET security

Το .NET Code Access Security βασίζεται σε assemblies και σε evidence. Evidence μπορεί να είναι οτιδήποτε συνάγεται από το assembly, αλλά συνήθως δημιουργείται από την πηγή του assembly – είτε αν το assembly εγκαταστάθηκε από το internet, κάποιο intranet ή είναι εγκατεστημένο στο τοπικό μηχάνημα (εάν το assembly έγινε download από άλλο μηχάνημα θα αποθηκευθεί σε μια θέση sandbox εντός της GAC και ως εκ τούτου δεν αντιμετωπίζεται σαν να έχει εγκατασταθεί τοπικά).

Άδειες χρήσης δίνονται σε ολόκληρα assemblies, και ένα assembly μπορεί να καθορίσει τα ελάχιστα δικαιώματα που απαιτεί μέσω των προσαρμοσμένων χαρακτηριστικών (βλέπε .NET metadata).

Όταν φορτώσει το assembly, το CLR θα χρησιμοποιήσει evidence για το assembly για να δημιουργήσει μια ομάδα αδειών χρήσης με έναν η περισσότερους κωδικούς δικαιωμάτων πρόσβασης.

Το CLR στην συνέχεια θα κάνει έλεγχο για να βεβαιωθεί ότι αυτή η ομάδα αδειών χρήσης, περιέχει τα απαιτούμενα δικαιώματα που καθορίζονται από το assembly.

Το .NET code μπορεί να εκτελέσει μια εντολή για πρόσβαση σε κωδικό ασφαλείας εάν του ζητηθεί. Αυτό σημαίνει ότι ο κώδικας κάποια προνομιακή δράση μόνο εάν όλα τα assemblies όλων των μεθόδων έχουν την συγκεκριμένη άδεια. Εάν ένα assembly δεν έχει τη συγκεκριμένη άδεια η εξαίρεση ασφάλειας απορρίπτεται.

Το .NET code μπορεί επίσης να εκτελέσει Linked Demand για να πάρει την άδεια από το call stack. Σε αυτή την περίπτωση το CLR θα εξετάσει για μια μόνο μέθοδο στην call stack στη θέση TOP να έχει την καθορισμένη άδεια. Εδώ η στοίβα-πέρασμα δεσμεύεται σε μια μέθοδο στην call stack με την οποία το CLR υποθέτει ότι όλες οι άλλες μέθοδοι στην CALL STACK έχουν την καθορισμένη άδεια.

1.2.4 Satellite assemblies

Σε γενικές γραμμές, τα assemblies πρέπει να περιλαμβάνουν culture-neutral πόρους. Εάν θέλουμε να εντοπίσουμε την assembly (για παράδειγμα, χρήση διαφορετικών strings για διαφορετικά locales) θα πρέπει να χρησιμοποιήσουμε δορυφόρους assemblies – ειδικά, resource-only assemblies. Όπως υποδηλώνει και το όνομα, ένας δορυφόρος που σχετίζεται με ένα assembly, καλείται κύριο assembly. Αυτό το assembly (lib.dll) θα περιέχει τα neutral-resources. Κάθε δορυφόρος έχει το όνομα της βιβλιοθήκης που συνδέεται με .resources (για παράδειγμα lib.resources.dll).

Στο δορυφόρο δίνεται ένα non-neutral culture όνομα, αλλά δεδομένου ότι αυτό αγνοείται από τα υφιστάμενα συστήματα αρχείων των Windows(FAT32 και NTFS) αυτό θα σήμαινε ότι θα μπορούσαν να υπάρχουν πολλά αρχεία με το ίδιο όνομα PE σε ένα φάκελο. Δεδομένου ότι αυτό δεν είναι εφικτό, οι δορυφόροι πρέπει να αποθηκεύονται σε υποφακέλους κάτω από το φάκελο της εφαρμογής.

Για παράδειγμα, ένας δορυφόρος με UK English resources θα έχει όνομα .NET όπως: "lib.resources Version=0.0.0.0 Culture=en-GB PublicKeyToken=null", ένα PE όνομα

αρχείου όπως lib.resources.dll και θα είναι αποθηκευμένο σε έναν υποφάκελο που ονομάζεται en-GB.

Οι δορυφόροι φορτώνονται από ένα .NET class το οποίο καλείται System.Resources.ResourceManager. Ο προγραμματιστής οφείλει να παρέχει το όνομα του πόρου και πληροφορίες σχετικά με το κύριο assembly (με neutral resources). Το ResourceManager class θα διαβάσει τις τοπικές ρυθμίσεις της μηχανής και θα τις χρησιμοποιήσει μαζί με το όνομα του βασικού assembly για να λάβει το όνομα του δορυφόρου και το όνομα του υποφακέλου που περιέχεται σε αυτόν. Το ResourceManager στη συνέχεια μπορεί να φορτώσει τον δορυφόρο και να εντοπίσει την τοπική πηγή.

1.2.5 Αναφορά σε assemblies

Υπάρχει η δυνατότητα να γίνει αναφορά σε ένα εκτελέσιμο κώδικα βιβλιοθήκης με τη χρήση του /reference flag του μεταγλωττιστή της C#.

Σε κοινά assemblies είναι αναγκαίο να δοθούν ισχυρά ονόματα ώστε να προσδιορίζεται η μοναδικότητα του assembly το οποίο είναι πιθανό να κατανεμηθεί μεταξύ των applications. Η ισχυρή ονομασία αποτελείται από το δημόσιο κλειδί token, culture, version και το PE όνομα αρχείου. Εάν κάποιο assembly είναι πιθανό να χρησιμοποιηθεί για το σκοπό της ανάπτυξης που είναι shared assembly, η διαδικασία ισχυρής ονοματοδοσίας περιλαμβάνει μόνο δημιουργία δημόσιου κλειδιού. Το ιδιωτικό κλειδί δεν δημιουργείται εκείνη τη στιγμή. Δημιουργείται μόνο όταν το assembly έχει αναπτυχθεί.

1.2.6 Γλώσσα της assembly

Η assembly είναι χτισμένη με τον CIL κώδικα ο οποίος είναι μια ενδιάμεση γλώσσα. Το framework μετατρέπει εσωτερικώς τον CIL[bytecode] σε μητρικό κώδικα της assembly.

Εάν έχουμε ένα πρόγραμμα που τυπώνει “Hello World”, ο αντίστοιχος κώδικας με την μέθοδο του CIL είναι:

```
.method private hidebysig static void  Main(string[] args) cil managed {
 .entrypoint
 .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() = (01 00 00 00)
 // Code size       11 (0xb)
 .maxstack  1
 IL_0000:  ldstr      "Hello World"
 IL_0005:  call       void [mscorlib]System.Console::WriteLine(string)
 IL_000a:  ret } // end of method Class1::Main
```

Έτσι, ο CIL κώδικας φορτώνει το String στη στοίβα. Στη συνέχεια καλεί την λειτουργία Writeline και επιστρέφει.

1.3 Metadata (Μεταδεδομένα)

Εισαγωγή

Όλο το CIL αυτό-περιγράφεται μέσω του .NET Metadata. Το CLR ελέγχει τα μεταδεδομένα ώστε να διασφαλίσει ότι καλείται η σωστή διαδικασία. Τα μεταδεδομένα συνήθως δημιουργούνται από τους μεταγλωττιστές της γλώσσας αλλά οι προγραμματιστές μπορούν να δημιουργήσουν τα δικά τους μεταδεδομένα μέσω των προσαρμοσμένων χαρακτηριστικών.

Τα μεταδεδομένα περιέχουν πληροφορίες σχετικά με το assembly και χρησιμοποιούνται επίσης για την εκτέλεση αντανακλαστικών δυνατοτήτων προγραμματισμού του .NET Framework.

1.3.1 .NET metadata

Το .NET metadata στο Microsoft .NET Framework , αναφέρεται σε ορισμένες δομές δεδομένων ενσωματωμένες με τον CIL κώδικα που περιγράφει την υψηλού επιπέδου δομή του κώδικα. Τα μεταδεδομένα περιγράφουν όλες τις τάξεις και την

κατηγορία μελών που το παρών assembly θα καλέσει από ένα άλλο assembly. Τα μεταδεδομένα για μια μέθοδο περιέχουν την πλήρη περιγραφή της μεθόδου (και το assembly που περιέχει την τάξη) , το τύπο που επιστρέφει και όλες τις παραμέτρους της μεθόδου.

Ένας μεταγλωττιστής γλώσσας του .NET θα δημιουργήσει τα μεταδεδομένα και θα τα αποθηκεύσει στο assembly που περιλαμβάνει το CIL. Όταν το CLR εκτελέσει το CIL θα κάνει έλεγχο για να βεβαιωθεί ότι τα μεταδεδομένα της μεθόδου που καλείται, είναι ίδια με τα μεταδεδομένα που είναι αποθηκευμένα στη μέθοδο που καλείται. Με αυτό τον τρόπο εξασφαλίζεται ότι μια μέθοδος μπορεί να κληθεί με τον ακριβή αριθμό παραμέτρων και ακριβώς τους σωστούς τύπους παραμέτρων.

1.3.2 Γνωρίσματα

Οι προγραμματιστές μπορούν να προσθέσουν τα μεταδεδομένα στον κώδικά τους μέσω γνωρισμάτων. Υπάρχουν δύο είδη γνωρισμάτων , τα τυπικά και τα ψευδό-τυπικά γνωρίσματα και για τον προγραμματιστή έχουν την ίδια σύνταξη. Τα γνωρίσματα στον κώδικα είναι μηνύματα προς τον μεταγλωττιστή για τη δημιουργία μεταδεδομένων.

Στο CIL, μεταδεδομένα όπως inheritance modifiers, scope modifiers και οτιδήποτε δεν είναι κώδικας πράξης ή ροής, αναφέρονται ως γνωρίσματα.

Ένα σύνηθες γνώρισμα είναι μια κανονική κλάση που κληρονομείται από την κλάση Attribute. Ένα σύνηθες γνώρισμα μπορεί να χρησιμοποιηθεί σε οποιαδήποτε μέθοδο, ιδιότητα, κλάση ή σε ένα ολόκληρο assembly με την σύνταξη:

```
[AttributeName(optional parameter, optional name=value pairs)] , όπως στο

[Custom]
 [Custom(1)]
 [Custom(1, Comment="yes")]
```

Τα συνήθη γνωρίσματα χρησιμοποιούνται εκτενώς από το .NET Framework. Το Windows Communication Framework χρησιμοποιεί γνωρίσματα για να καθορίσει τις συμβάσεις παροχής υπηρεσιών, ενώ το ASP.NET τα χρησιμοποιεί για μεθόδους όπως υπηρεσίες web. Το LINQ και το SQL τα χρησιμοποιεί για να καθορίσει την χαρτογράφηση των κλάσεων στο underlying relational schema, το Visual studio τα χρησιμοποιεί για να συγκεντρώσει τις ιδιότητες ενός αντικειμένου, ο προγραμματιστής της κλάσης προσδιορίζει την κατηγορία κλάσης του αντικειμένου καλώντας το γνώρισμα [Category]. Τα συνήθη γνωρίσματα ερμηνεύονται από τον κώδικα της εφαρμογής και όχι από το CLR.

Όταν ο μεταγλωττιστής συναντήσει ένα σύνηθες γνώρισμα, θα δημιουργήσει συνήθη μεταδεδομένα που δεν αναγνωρίζονται από το CLR. Ο προγραμματιστής παρέχει το κώδικα ώστε να διαβαστούν τα μεταδεδομένα και να επεξεργαστούν.

Για παράδειγμα, το γνώρισμα που φαίνεται στο παράδειγμα μπορεί να χειριστεί από τον κώδικα:

```
class CustomAttribute : Attribute
{
  private int paramNumber = 0;
  private string comment = "";

  public CustomAttribute() { }
  public CustomAttribute(int num) { paramNumber = num; }

  public String Comment
  {
    set { comment = value; }
  }
}
```

Το όνομα της κλάσης αντιστοιχεί στο όνομα του γνωρίσματος. Ο μεταγλωττιστής της C# προσθέτει αυτόματα το string “Attribute” στο τέλος από κάθε όνομα γνωρίσματος. Κατά συνέπεια, κάθε κλάση γνωρίσματος θα πρέπει να τελειώνει με αυτό το string αλλά είναι επιτρεπτό να καθοριστεί ένα γνώρισμα χωρίς τη κατάληξη του γνωρίσματος. Όταν επισυνάπτουμε ένα γνώρισμα σε ένα στοιχείο, ο μεταγλωττιστής θα εξετάσει το κυριολεκτικό όνομα και το όνομα όπου το γνώρισμα έχει προστεθεί στο τέλος, π.χ. εάν πρόκειται να γράψουμε [Custom] ο μεταγλωττιστής θα ελέγξει για το Custom και το CustomAttribute. Εάν υπάρχουν και τα δύο, ο μεταγλωττιστής αποτυγχάνει.

Το γνώρισμα μπορεί να διορθωθεί με “@” εάν δεν θέλουμε να διακινδυνεύσουμε ασάφεια, επομένως εάν γράψουμε [@Custom] δε θα ταιριάζει με το CustomAttribute. Όταν χρησιμοποιούμε το γνώρισμα επικαλείται το κατασκευαστή της κλάσης. Υποστηρίζονται overloaded κατασκευαστές. Τα ζεύγη name-value αντιστοιχίζονται με τις ιδιότητες , το όνομα δηλώνει το όνομα της ιδιοκτησίας και η αξία που παρέχεται καθορίζεται από την ιδιοκτησία.

Μερικές φορές υπάρχει ασάφεια σχετικά με το που επισυνάπτουμε το γνώρισμα.

Ας εξετάσουμε τον επόμενο κώδικα:

```
[Orange]
public int ExampleMethod(string input)
{
   //method body goes here
}
```

Τι είναι όμως αυτό που έχει επισημανθεί ως [Orange]?

Είναι η ExampleMethod, η τιμή επιστροφής ή ίσως και ολόκληρο το assembly?

Σε αυτή τη περίπτωση, ο μεταγλωττιστής θα προεπιλεγεί και θα αντιμετωπίσει το γνώρισμα σαν να έχει επισυναφθεί με τη μέθοδο. Εάν αυτός δεν ήταν ο σκοπός ή αν ο

δημιουργός επιθυμεί να διευκρινίσει τον κωδικό, ένα attribute target μπορεί να διευκρινιστεί. Γράφοντας [return: Orange] θα επιλέξει τη τιμή επιστροφής ως πορτοκαλί , [assembly: Orange] θα επιλέξει ολόκληρο το assembly. Οι έγκυροι στόχοι είναι assembly, πεδίο, event, μέθοδος, module, param, property , επιστροφή και τύπος.

Ένα ψευδό-τυπικό γνώρισμα χρησιμοποιείται σαν τα τυπικά γνωρίσματα αλλά δεν έχει ένα custom handler. Ο μεταγλωττιστής έχει εγγενή γνώση των γνωρισμάτων και χειρίζεται τον σημειωμένο κωδικό με διαφορετικά γνωρίσματα.

Γνωρίσματα όπως το Serializable και το Obsolete υλοποιούνται ως ψευδό-τυπικά γνωρίσματα. Τα ψευδό-τυπικά γνωρίσματα δε πρέπει να χρησιμοποιούνται από το ILASM , δεδομένου ότι έχει επαρκή σύνταξη για να περιγράψει τα μεταδεδομένα.

1.3.3 Αποθήκευση Μεταδεδομένων

Τα assemblies περιέχουν πίνακες με μεταδεδομένα. Αυτοί οι πίνακες περιγράφονται από τις προδιαγραφές του CIL. Οι πίνακες μεταδεδομένων θα έχουν καμία ή περισσότερες εισόδους και η θέση μιας εισόδου προσδιορίζει το δείκτη της. Όταν ο κώδικας CIL χρησιμοποιεί μεταδεδομένα το πράττει μέσω ενός token μεταδεδομένων. Αυτό είναι μια τιμή 32-bit όπου τα πρώτα 8 bits θα προσδιορίζουν τον κατάλληλο πίνακα μεταδεδομένων και τα υπόλοιπα 24bits θα δίνουν το δείκτη των μεταδεδομένων στον πίνακα. Το Framework SDK περιλαμβάνει ένα δείγμα που καλείται metainfo και το οποίο θα καταγράψει του πίνακες μεταδεδομένων σε ένα assembly, ωστόσο αυτές οι πληροφορίες είναι σπάνια χρήσιμες για έναν προγραμματιστή. Τα μεταδεδομένα σε ένα assembly μπορούν να προβληθούν χρησιμοποιώντας το εργαλείο ILDASM που παρέχεται από το .NET Framework SDK.

Reflection

Το reflection είναι η χρησιμοποίηση του API για να διαβάσει μεταδεδομένα του .NET. Το reflection API παρέχει μια λογική άποψη των μεταδεδομένων και όχι την κυριολεκτική άποψη που παρέχεται από εργαλεία όπως το metainfo. Το reflection

στην έκδοση 1.1 του .NET Framework μπορεί να χρησιμοποιηθεί για να επιθεωρήσει τις περιγραφές των κλάσεων και των μελών τους και επικαλείται μεθόδους. Ωστόσο δεν επιτρέπει την πρόσβαση στο χρόνο εκτέλεσης του CIL για μια μέθοδο. Στην έκδοση 2.0 του framework επιτρέπεται στο CIL για μια μέθοδο που θα ληφθεί. Άλλα εργαλεία μεταδεδομένων

Εκτός από το namespace System.Reflection , υπάρχουν και άλλα εργαλεία που είναι επίσης διαθέσιμα και μπορούν να χρησιμοποιηθούν για το χειρισμό των μεταδεδομένων. Το Microsoft .NET Framework φορτώνει μεταδεδομένα της CLR που χειρίζονται την βιβλιοθήκη και εκτελούνται σε μητρικό κώδικα. Επίσης μπορούν να χρησιμοποιηθούν εργαλεία για την ανάκτηση και τον χειρισμό των μεταδεδομένων που περιλαμβάνουν το PostSharp και το Mono Cecil.

1.4 Ασφάλεια

Το .NET έχει το δικό του μηχανισμό ασφαλείας με 2 γενικά χαρακτηριστικά:
Τον Κωδικό Ασφάλειας Πρόσβασης (CAS) και την επικύρωση με την επαλήθευση.
Ο κωδικός ασφάλειας πρόσβασης βασίζεται σε evidence τα οποία σχετίζονται με ένα συγκεκριμένο assembly. Συνήθως τα evidence είναι η πηγή του assembly(είτε είναι εγκατεστημένο στον τοπικό υπολογιστή είτε έχει εγκατασταθεί μέσω του intranet ή του internet). Ο κωδικός ασφάλειας πρόσβασης χρησιμοποιεί τα evidence για να καθορίσει τα δικαιώματα που χορηγούνται στον κώδικα. Άλλος κώδικας μπορεί να απαιτήσει χορήγηση συγκεκριμένης άδειας για το κώδικα που καλείται. Η κλήση χορήγησης άδειας προκαλεί το CLR να εκτελέσει μια call stack walk: κάθε assembly της κάθε μεθόδου στην call stack ελέγχεται για την απαιτούμενη άδεια. Εάν σε κάποιο assembly δεν παρέχεται η άδεια, a security exception is thrown.

Όταν ένα assembly έχει φορτωθεί , το CLR εκτελεί διάφορες δοκιμές. Δύο από αυτές τις δοκιμές είναι η επικύρωση και η επαλήθευση. Κατά την επικύρωση, το CLR ελέγχει εάν το assembly περιέχει έγκυρα μεταδεδομένα και CIL και εάν οι εσωτερικοί πίνακες είναι σωστοί.

Η επαλήθευση δεν είναι τόσο ακριβής. Οι μηχανισμοί επαλήθευσης ελέγχουν να δουν μήπως ο κώδικας κάνει κάτι που δεν είναι ‘ασφαλές’. Ο αλγόριθμος που

χρησιμοποιείται είναι αρκετά συντηρητικός. Ως εκ τούτου, περιστασιακοί κώδικες που δεν είναι ‘ασφαλείς’ δεν περνούν. Επισφαλείς κώδικες θα εκτελεστούν μόνο εάν το assembly έχει την άδεια “skip verification”, που σημαίνει σε γενικές γραμμές ότι είναι εγκατεστημένο στον τοπικό υπολογιστή.

Το .NET Framework χρησιμοποιεί τα AppDomains ως ένα μηχανισμό για την απομόνωση του code running κατά την διάρκεια μιας διαδικασίας. Τα AppDomains μπορούν να δημιουργηθούν και ένας κωδικός να φορτωθεί σε αυτά η να εκφορτωθεί από αυτά ανεξάρτητα από άλλα AppDomains. Αυτό αυξάνει την ανοχή σε σφάλματα της εφαρμογής, καθώς τα σφάλματα η τα crashes σε κάποιο AppDomains δεν επηρεάζουν τ υπόλοιπο κομμάτι της εφαρμογής. Τα AppDomains μπορούν επίσης να ρυθμιστούν ανεξάρτητα, με διαφορετικά δικαιώματα προστασίας. Αυτό μπορεί να συμβάλει στην αύξηση της ασφάλειας της εφαρμογής, απομονώνοντας ανασφαλείς κώδικες. Ο προγραμματιστής ωστόσο πρέπει να διαχωρίσει την εφαρμογή σε επιμέρους τομείς καθώς αυτό δε μπορεί να γίνει από το CLR.

1.5 Βιβλιοθήκη Κλάσεων

Ονοματοδοσία στην BCL

System

System. CodeDom

System. Collections

System. Diagnostics

System. Globalization

System. IO

System. Resources

System. Text

System. Text.RegularExpressions

Το .NET Framework περιλαμβάνει ένα σύνολο βιβλιοθήκης κλάσεων. Η βιβλιοθήκη κλάσης είναι οργανωμένη με μια ιεραρχία ονομάτων. Τα περισσότερα από αυτά που έχουν δημιουργηθεί στο APIs αποτελούν μέρος είτε του System.* είτε

της Microsoft.* ονοματοδοσίας. Αυτές οι βιβλιοθήκες κλάσης εφαρμόζουν έναν μεγάλο αριθμό κοινών λειτουργιών όπως είναι μεταξύ άλλων η ανάγνωση αρχείου και εγγραφή σε αυτά, γραφική απόδοση, αλληλεπίδραση βάσης δεδομένων καθώς και χειρισμό XML εγγράφων.

Οι .NET βιβλιοθήκες κλάσης είναι διαθέσιμες σε όλες τις γλώσσες που ακολουθούν την CLI. Η .NET βιβλιοθήκη κλάσης χωρίζεται σε δύο μέρη: την Base Class Library και την Framework Class Library.

Η Base Class Library (BCL) περιλαμβάνει ένα μικρό υποσύνολο από τη συνολική βιβλιοθήκη κλάσης και αποτελεί τον πυρήνα των κλάσεων που χρησιμεύουν ως βασικοί API της CLR. Οι κλάσεις στο mscorlib.dll και μερικές κλάσεις στο System.dll και στο System.core.dll θεωρούνται μέρος της BCL. Οι κλάσεις BCL είναι διαθέσιμες και στα δυο .NET Framework καθώς και τις εναλλακτικές υλοποιήσεις του όπως το .NET Compact Framework, Microsoft Silverlight και το Mono.

Το Framework Class Library (FCL) είναι ένα υπερσύνολο των BCL κλάσεων και αναφέρεται στο σύνολο της βιβλιοθήκης κλάσης που φορτώνεται στο .NET Framework. Περιλαμβάνει ένα εκτεταμένο σύνολο βιβλιοθηκών, συμπεριλαμβανομένων των WinForms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation ανάμεσα από άλλα. Το FCL είναι πολύ μεγαλύτερο σε πεδία εφαρμογής από τις τυπικές βιβλιοθήκες για γλώσσες όπως την C++ και συγκρίσιμο με τα πεδία εφαρμογής των βιβλιοθηκών της Java.

1.6 Διαχείριση Μνήμης

Το CLR του .NET Framework απελευθερώνει το προγραμματιστή από το βάρος της διαχείρισης μνήμης(κατανομή και απελευθέρωση όταν ολοκληρωθεί) καθώς τα πραγματοποιεί η ίδια η διαχείριση μνήμης. Για το σκοπό αυτό, η μνήμη που διατίθεται για συγκεκριμένους .NET τύπους (αντικείμενα), πραγματοποιείται συνεχώς από τη διαχείριση heap, μια δεξαμενή μνήμης από την CLR. Όσο υπάρχει μια αναφορά σε ένα αντικείμενο το οποίο θα μπορούσε να είναι είτε άμεση αναφορά σε ένα αντικείμενο είτε μέσω ενός γραφήματος αντικειμένων, το αντικείμενο

θεωρείται ότι είναι σε χρήση από την CLR. Όταν δεν υπάρχει αναφορά σε κάποιο αντικείμενο και δε μπορεί να προσπελαστεί ή να χρησιμοποιηθεί, αχρηστεύεται. Ωστόσο, εξακολουθεί να παραμένει στη μνήμη που έχει καταχωρηθεί. Το .NET Framework περιλαμβάνει ένα συλλέκτη αχρηστευμένων αντικειμένων ο οποίος εκτελείται περιοδικά σε ξεχωριστό τμήμα από αυτό της εφαρμογής, απαριθμώντας όλα τα αχρηστευμένα αντικείμενα και ανακτώντας τη μνήμη που τους έχει κατανεμηθεί.

Το .NET Garbage Collector (GC) είναι ένας μη ντετερμινιστικός, συμπιεσμένος, mark-and-sweep συλλέκτης αχρηστευμένων αντικειμένων. Το GC τρέχει μόνο όταν ένας συγκεκριμένος αριθμός μνήμης έχει χρησιμοποιηθεί ή υπάρχει αρκετή πίεση για τη μνήμη στο σύστημα. Επειδή δεν είναι εγγυημένο, όταν οι συνθήκες για την διεκδίκηση εκ νέου της μνήμης έχουν εξαντληθεί, η εκτέλεση του GC είναι μη-ντετερμινιστική. Κάθε εφαρμογή του .NET έχει ένα σύνολο από ρίζες οι οποίες είναι δείκτες αντικειμένων σχετικά με τη διαχείριση των αντικειμένων. Αυτές περιλαμβάνουν αναφορές σε στατικά αντικείμενα και αντικείμενα που ορίζονται ως τοπικές μεταβλητές ή μεθόδους παραμέτρων επί του παρόντος στο πεδίο εφαρμογής καθώς και αντικείμενα που αναφέρονται από τα μητρώα της CPU. Όταν το GC εκτελείται, σταματά προσωρινά την εφαρμογή και για κάθε αντικείμενο που αναφέρεται στην ρίζα απαριθμεί αναδρομικά όλα τα αντικείμενα που είναι προσβάσιμα από τα αντικείμενα της ρίζας και τα αναφέρει ως προσβάσιμα.

Χρησιμοποιεί μεταδεδομένα .NET και reflection για να ανακαλύψει τα έγκλειστα αντικείμενα από ένα αντικείμενο και στη συνέχεια κατ' επανάληψη walk them. Στη συνέχεια απαριθμεί όλα τα αντικείμενα του σωρού (που αρχικά είχαν διατεθεί συνεχόμενα) χρησιμοποιώντας reflection. Όλα τα αντικείμενα που δεν χαρακτηρίζονται ως προσβάσιμα, αχρηστεύονται. Αυτή είναι η mark phase. Δεδομένου ότι η μνήμη που κατέχουν τα αχρηστευμένα αντικείμενα δεν έχει καμία συνέπεια, θεωρείται ελεύθερος χώρος. Ωστόσο αυτό αφήνει κομμάτια ελεύθερου χώρου μεταξύ των αντικειμένων που αρχικά ήταν συνεχόμενα. Τα αντικείμενα στη συνέχεια συμπιέζονται από κοινού με τη χρήση memcpy να τα αντιγράψει στον ελεύθερο χώρο για να γίνουν συνεχόμενα ξανά. Κάθε αναφορά σε ένα αντικείμενο αναιρείται με τη μετακίνηση του αντικειμένου ανανεώνεται ώστε να αντικατοπτρίζει

τη νέα θέση από το GC. Η εφαρμογή συνεχίζεται αφού η συλλογή των απορριμμάτων έχει ολοκληρωθεί.

Το GC που χρησιμοποιείται από το .NET Framework είναι στη πραγματικότητα generational. Τα αντικείμενα ανατίθενται σε μια generation. Νεοδημιουργηθείσα αντικείμενα ανήκουν στην Generation 0. Τα αντικείμενα που επιβιώνουν από ένα ξεσκαρτάρισμα αχρηστευμένων αντικειμένων μεταφέρονται στη κατηγορία Generation 1 και αυτά που θα επιβιώσουν από άλλο ένα ακόμα ξεσκαρτάρισμα είναι Generation 2 αντικείμενα. Το .NET Framework χρησιμοποιεί έως Generation 2 αντικείμενα. Μεγαλύτερης γενιάς αντικείμενα συλλέγονται λιγότερο ως αχρηστευμένα απ' ότι τα αντικείμενα χαμηλότερης γενιάς. Αυτό βοηθά στην αύξηση της αποτελεσματικότητας της συλλογής απορριμμάτων καθώς τα παλαιότερα αντικείμενα τείνουν να έχουν μεγαλύτερη διάρκεια ζωής από τα νεότερα. Έτσι με την αφαίρεση των παλαιοτέρων (και συνεπώς πιο πιθανό να επιζήσουν από κάποιο ξεσκαρτάρισμα) αντικειμένων από το πεδίο εφαρμογής ενός κύκλου συλλογής , λιγότερα αντικείμενα χρειάζεται να ελεγχθούν και να συμπιεστούν.

Κεφάλαιο 2

Java vs. .NET Framework

Εισαγωγή

Το CLI και οι γλώσσες .NET όπως π.χ. η C# και η Visual Basic έχουν πολλές ομοιότητες με τις Java και JVM της Sun. Και οι δυο βασίζονται σε ένα εικονικό μοντέλο μηχανής που υποκρύπτει τις λεπτομέρειες από το hardware του υπολογιστή πάνω στο οποίο τρέχουν τα προγράμματά τους. Και οι δυο χρησιμοποιούν το δικό τους ενδιάμεσο κώδικα byte. Η Microsoft το ονομάζει Common Intermediate Language (CIL) ενώ η Sun το ονομάζει Java bytecode.

Στο .NET ο byte-code καταρτίζεται πάντα πριν από την εκτέλεση, είτε Just In Time (JIT), είτε πριν απ την εκτέλεση χρησιμοποιώντας τη δυνατότητα Native Image Generator (NGEN).

Στη Java το byte-code ερμηνεύεται είτε καταρτισμένο εκ των προτέρων, είτε με τη μέθοδο JIT. Και τα δυο παρέχουν εκτενείς βιβλιοθήκες που αντιμετωπίζουν πολλές κοινές απαιτήσεις προγραμματισμού και πολλά ζητήματα ασφαλείας που παρουσιάζονται με άλλη προσέγγιση. Η ονοματοδοσία που παρέχεται στο .NET Framework μοιάζει αρκετά με τα πακέτα πλατφόρμας στη προδιαγραφή Java EE API.

Το .NET υπό πλήρη μορφή μπορεί να εγκατασταθεί μόνο σε υπολογιστές που τρέχουν σε λειτουργικό σύστημα Microsoft Windows ενώ η Java στο σύνολο της μπορεί να εγκατασταθεί σε υπολογιστές που τρέχουν σε μια ποικιλία λειτουργικών συστημάτων όπως Linux, Solaris, Mac OS η Windows. Από την αρχή το .NET έχει υποστηρίξει πολλές γλώσσες προγραμματισμού και στον πυρήνα της παραμένει τυποποιημένη κατά τέτοιο τρόπο ώστε να μπορεί να εφαρμοστεί και σε άλλες πλατφόρμες (παρόλο που η εφαρμογή της Microsoft ασχολείται αποκλειστικά με Windows, Windows CE και Xbox πλατφόρμες). Η Java Virtual Machine σχεδιάστηκε

επίσης για να είναι όχι μόνο γλώσσα αλλά και λειτουργικό σύστημα και λανσαρίστηκε με το σύνθημα «Γράψε μια φορά, τρέξε οπουδήποτε» . Ενώ η Java παρέμενε για καιρό η πλέον χρησιμοποιούμενη γλώσσα για το JVM με μεγάλη διαφορά, η πρόσφατη υποστήριξη δυναμικών γλωσσών έχει αυξήσει την δημοτικότητα των εναλλακτικών λύσεων. Κάποιες από αυτές είναι η JRuby, Scala και Groovy.

Η εφαρμογή αναφοράς Java της Sun (συμπεριλαμβανομένης και της βιβλιοθήκης κλάσης, τον μεταγλωττιστή, την virtual machine και τα διάφορα εργαλεία που σχετίζονται με τη πλατφόρμα Java) είναι ανοιχτού κώδικα υπό την άδεια χρήσης GNU GPL με εξαίρεση το Classpath. Ο πηγαίος κώδικας για τη βιβλιοθήκη βάσης κλάσεων του .NET Framework είναι διαθέσιμος μόνο για σκοπούς αναφοράς στα πλαίσια της Microsoft Reference License.

Το Mono Project, που υποστηρίζεται από τη Novell, έχει αναπτύξει μια εφαρμογή ανοιχτού κώδικα των προτύπων ECMA που αποτελεί μέρος του .NET Framework, όπως και οι περισσότερες μη τυποποιημένες ECMA βιβλιοθήκες στο .NET της Microsoft. Η εφαρμογή Mono είναι σχεδιασμένη να τρέχει σε Linux, Solaris, Mac OS X, BSD, HP-UX και Windows πλατφόρμες. Η Mono περιλαμβάνει τη CLR, βιβλιοθήκες κλάσης και μεταγλωττιστές για την C# και την VB.NET. Η τρέχουσα έκδοση υποστηρίζει όλα τα APIs στην έκδοση 2.0 της Microsoft .NET. Πλήρης υποστήριξη υπάρχει για την C# 3.0 LINQ to Objects και LINQ to XML.

2.1 Νομικά ζητήματα

2.1.1 Τυποποίηση

Οι δυο πλατφόρμες, οι βιβλιοθήκες προγραμματισμού τους, οι δυαδικές τους μορφές και το runtime περιβάλλον τους, διέπονται σε μεγάλο βαθμό από πολύ διαφορετικές έννοιες. Διεθνείς οργανισμοί τυποποίησης, η ECMA International και το ISO/IEC καθορίζουν τα πρότυπα για το εκτελέσιμο περιβάλλον του .NET (γνωστό

και ως Common Language Infrastructure, η CLI) και το εκτελέσιμο σχήμα του .NET(γνωστό και ως Common Intermediate Language, η CIL) εξαιρουμένων όμως των περισσοτέρων θεμελιωδών κλάσεων (Base Class Library, η BCL). Αυτή η επίσημη επιτροπή τυποποίησης είναι σύμφωνη με το τρόπο που πολύ δημοφιλείς γλώσσες όπως η COBOL, Fortran και C έχουν τυποποιηθεί κατά το παρελθόν. Τα πρότυπα δεν περιλαμβάνουν πολλές νέες βιβλιοθήκες που η Microsoft έχει υλοποιήσει πάνω στο τυπικό framework, όπως εκείνες για πρόσβαση σε βάσεις δεδομένων ή δημιουργία GUI και τις εφαρμογές Web, όπως τα Windows Forms, ASP.NET και ADO.NET.

Μέχρι σήμερα, κανένα τμήμα της Java δεν έχει τυποποιηθεί από την Ecma International, το ISO/IEC, το ANSI ή οποιοδήποτε άλλο οργανισμό προτύπων. Ενώ η Sun Microsystems έχει απεριόριστα και αποκλειστικά νομικά δικαιώματα να τροποποιήσει τα εμπορικά σήματά της, η Sun εθελοντικά συμμετέχει σε μια διαδικασία που ονομάζεται Java Community Process (JCP) που επιτρέπει στους ενδιαφερομένους να προτείνουν αλλαγές σε οποιαδήποτε τεχνολογία της Java (από τη γλώσσα και τα εργαλεία μέχρι τα API) μέσω φόρουμ, διαβουλεύσεις και ομάδες εμπειρογνωμόνων. Η JPC απαιτεί συνδρομή μέλους για τους εμπορικούς συνεργάτες, ενώ οι μη εμπορικοί συνεργάτες και ιδιώτες μπορούν να συνεισφέρουν δωρεάν.

Στο πλαίσιο των κανόνων του JPC, ο καθένας μπορεί να υποβάλλει πρόταση για νέα Platform Edition Specifications η να προτείνει αλλαγές στην γλώσσα Java. Όλες οι προτάσεις επανεξετάζονται και ψηφίζονται από τα ενδιαφερόμενα μέλη της JPC σε διάφορα στάδια και σε ολόκληρο το lifecycle τους. Ωστόσο όταν πρόκειται να συμπεριληφθούν μεταβολές στο επίπεδο αναφοράς εφαρμογών (Java SE, Java EE και Java ME) οι τροποποιήσεις μπορεί να απορριφθούν από την Sun που διατηρεί την απόλυτη δύναμη του βέτο.

Τα πρότυπα Java επεξεργάζονται από μια σουίτα δοκιμής εφαρμογών που εξετάζουν κάθε πτυχή μιας συγκεκριμένης εφαρμογής της Java έναντι αυστηρών προδιαγραφών. Μόνο αν μια εφαρμογή καταφέρει να περάσει τις δεκάδες χιλιάδες μεμονωμένων δοκιμών μπορεί να χρησιμοποιήσει το εμπορικό σήμα “Java” καθώς και τα σχετικά λογότυπα και εμπορικά σήματα.

2.1.2 Άδεια

Java

Ενώ το “Java” αποτελεί σήμα κατατεθέν της Sun και μόνο η Sun μπορεί να χορηγήσει άδεια εκμετάλλευσης του ονόματος “Java”, υφίστανται πολυάριθμα ελεύθερα λογισμικά τα οποία είναι συμβατά με τη Sun Java. Πιο συγκεκριμένα, το GNU Class path και το GCJ παρέχουν ένα ελεύθερο λογισμικό βιβλιοθήκης κλάσεων και ένα μεταγλωττιστή που είναι εν μέρει συμβατοί με την τρέχουσα έκδοση του Sun Java. Η Sun ανακοίνωσε στις 14 Νοεμβρίου 2006 ότι όλοι οι πηγαίοι κώδικες Java, με εξαίρεση το κώδικα κλειστού τύπου για τον οποίο δεν διατηρεί τα δικαιώματα, θα κυκλοφορήσουν σε τροποποιημένη εκδοχή της GPL και κυκλοφόρησε δυο βασικά μέρη του JRE και του JDK. Το Hotspot και το μεταγλωττιστή Java υπό την GPL.

Μετά από δέσμευσή τους, η Sun κυκλοφόρησε το πλήρη πηγαίο κώδικα της βιβλιοθήκης κλάσης σύμφωνα με την GPL στις 8 Μαΐου 2007, με εξαίρεση ορισμένα κομμάτια που είχαν πάρει την άδεια της Sun από τρίτους και οι οποίοι δεν ήθελαν τον κωδικό τους να κυκλοφορήσει με open-source άδεια. Στόχος της Sun ήταν να αντικαταστήσει τα τμήματα τα οποία παρέμεναν κλειστά με εναλλακτικές υλοποιήσεις και να κάνει την βιβλιοθήκη κλάσης εντελώς ανοικτή. Τον Ιούνιο του 2008, η Red Hat ανακοίνωσε ότι το IcedTea project είχε περάσει την αυστηρή προδιαγραφή της Java TCK, υποδηλώνοντας μια πλήρως λειτουργική υλοποίηση ανοικτού πηγαίου κώδικα της πλατφόρμας Java.

.NET

Το εκτελέσιμο περιβάλλον CLI του Microsoft .NET καθώς και κάποιες από τις αντίστοιχες βιβλιοθήκες κλάσης, έχουν τυποποιηθεί και μπορούν ελεύθερα να εφαρμοστούν χωρίς άδεια. Ελάχιστα πρότυπα ελεύθερου λογισμικού έχουν υλοποιηθεί, όπως το Mono Project και το DotGNU. Το Mono Project έχει επίσης εφαρμόσει πολλές από τις μη τυποποιημένες βιβλιοθήκες, εξετάζοντας εργαλεία της Microsoft, παρόμοια με το GNU Classpath και της Java. Η Microsoft σήμερα διανέμει μια κοινή έκδοση από το περιβάλλον runtime του .NET για ακαδημαϊκή

χρήση, ωστόσο υποστηρίζεται μόνο στα Windows XP SP2 και δεν έχει ενημερωθεί μετά το .NET 2.0.

Το Mono project αποσκοπεί στο να αποφευχθεί παραβίαση για ευρεσιτεχνίες ή πνευματικά δικαιώματα και στο βαθμό που αυτές είναι επιτυχής, το project μπορεί με ασφάλεια να διανεμηθεί και να χρησιμοποιηθεί υπό την GPL. Στις 2 Νοεμβρίου 2006, η Microsoft και η Novell ανακοίνωσαν μια κοινή συμφωνία με την οποία η Microsoft υποσχέθηκε να μην μηνύσει την Novell η τους πελάτες της για παραβίαση ευρεσιτεχνίας. Σύμφωνα με δήλωση στο blog του πρωτεργάτη του Mono project, Miguel de Icaza, αυτή η συμφωνία αφορά αποκλειστικά το Mono για προγραμματιστές και χρήστες της Novell. Λόγω της πιθανής απειλής των ευρεσιτεχνιών της Microsoft , το FSF συνιστά να αποφεύγεται η δημιουργία λογισμικού το οποίο εξαρτάται από την Mono η την C#.

Η συμφωνία Microsoft/Novell επικρίθηκε από ορισμένους στην κοινότητα ανοικτής πηγής διότι παραβιάζει τις αρχές ίσων δικαιωμάτων σε όλους τους χρήστες ενός συγκεκριμένου προγράμματος. Σε απάντηση της συμφωνίας Microsoft/Novell, η Free Software Foundation αναθεώρησε το General Public License του GNU ώστε να κλείσει το κενό που χρησιμοποιήθηκε από τη Microsoft και τη Novell να παρακάμψουν τις πολύ περιοριστικές διατάξεις της GPL σχετικά με τις συμφωνίες πατεντών. Το FSF δήλωσε επίσης ότι με την πώληση κουπονιών για λογισμικό Linux από την Novell, -μηχανισμός με τον οποίο η Microsoft καταστρατήγησε την άδεια GNU- θεωρείται ότι η Microsoft είναι ο πωλητής του Linux και ως εκ τούτου υπόκεινται πλήρως στους όρους και τις προϋποθέσεις που ορίζονται από το GPL.

2.2 Κοινότητα

Η Sun, ιδιοκτήτρια της Java, εργάζεται με ανοιχτή κουλτούρα, επιτρέποντας σε άτομα από οργανώσεις έως ιδιώτες, να κατευθύνουν τη διαδικασία λήψης αποφάσεων. Η Sun διατηρεί αποκλειστικά και απεριόριστα νομικά δικαιώματα στις ιδιότητες της Java και η κοινότητα της Java συμμορφώνεται σε αυτά τα δικαιώματα.

Η Java έχει αναπτυχθεί σε δημοτικότητα για να γίνει μια από τις πιο δημοφιλείς γλώσσες στις αρχές του 21^{ου} αιώνα και ο πλουραλιστικός χαρακτήρας της ανάπτυξης της έχει οδηγήσει σε πολλές διαφορετικές ομάδες την αντιμετώπιση των ίδιων (η παρόμοιων) προβλημάτων. Το θέμα αυτό είναι χαρακτηριστικό στο χώρο των επιχειρήσεων(web/Ajax/Web2.0 εφαρμογές), όπου δεν πρέπει κάποιος να είναι εξοικειωμένος μόνο με την Java αλλά και με διάφορα ανταγωνιστικά frameworks.

Ενώ η Microsoft έχει αναπτύξει την C# και το .NET χωρίς ένα επίσημο σύστημα κοινοτικής συνεισφοράς, η γλώσσα και ορισμένα τμήματα της εκτελέσιμης μορφής και χρόνου εκτέλεσης έχουν τυποποιηθεί και διανεμηθεί δωρεάν μέσω της Ecma και του ISO σε μια ανοικτή και ουδέτερη διαδικασία και όχι μια διαδικασία που διατηρεί το βέτο και τα δικαιώματα χρήσης για τη Microsoft. Ωστόσο τα πρότυπα δεν περιλαμβάνουν πολλές νέες βιβλιοθήκες που η Microsoft έχει υλοποιήσει πάνω στο τυπικό framework. Πολυάριθμα projects λογισμικού της C# και CLI σε κοινότητες, help sections, site τεκμηρίωσης και φόρουμ συζητήσεων είναι ενεργά για την ανάπτυξη και συντήρηση, συμπεριλαμβανομένων εκείνων που αφορούν την ανάπτυξη των Windows με το Microsoft .NET η το Mono Project, δωρεάν λογισμικό ανάπτυξης λειτουργικών συστημάτων από το Mono Project και mobile ανάπτυξη με τη χρήση του συμπαγούς framework του Microsoft . NET.

Η Microsoft διανέμει μια κοινή πηγή (version 1.0) του .NET virtual machine που μπορεί να καταρτίζεται και να χρησιμοποιείται στα Windows, FreeBSD, Mac OS X και σε άλλες πλατφόρμες. Μια ενημερωμένη έκδοση (2.0) είναι διαθέσιμη σήμερα αλλά η μοναδική πλατφόρμα που την υποστηρίζει είναι αυτή των Windows.

2.3 Παραδοσιακές εφαρμογές υπολογιστών

2.3.1 Εφαρμογές Desktop

Παρόλο που το AWT (Abstract Windowing Toolkit) της Java και οι βιβλιοθήκες Swing δεν υπολείπονται χαρακτηριστικών, η Java έχει παλέψει για να αποκτήσει μια σταθερή θέση στην αγορά desktop. Η Sun Microsystems δεν έχει καταφέρει ,κατά την άποψη ορισμένων, να προωθήσει τη Java σε προγραμματιστές και τελικούς χρήστες με ένα τρόπο ώστε να τη καθιστά ελκυστική επιλογή για desktop λογισμικό. Ακόμα και τεχνολογίες όπως η Java Web Start, που έχουν λίγες ομοιότητες με αντίπαλες γλώσσες και πλατφόρμες, έχουν προωθηθεί ελάχιστα.

Η έκδοση της Java 6.0 στις 11 Δεκεμβρίου 2006, έδειξε μια επικέντρωση στην αγορά των desktop με μια εκτεταμένη σειρά νέων εργαλείων για στενότερη επαφή με το desktop. Το 2007 στη διάσκεψη JavaOne, η Sun έκανε περαιτέρω ανακοινώσεις σχετικά με το desktop συμπεριλαμβανομένης και μιας νέας γλώσσας με στόχο το Adobe Flash (JavaFX), ένα νέο και πιο ελαφρύ τρόπο στο κατέβασμα του JRE το οποίο έχει μειωθεί στα 2 mb καθώς και μια ανανεωμένη εστίαση στις βιβλιοθήκες πολυμέσων.

Μια εναλλακτική λύση στην AWT και την Swing είναι το Standard Widget Toolkit (SWT), η οποία αναπτύχτηκε αρχικά από την IBM και τώρα ανήκει στην Eclipse Foundation. Επιχειρεί να επιτυγχάνονται καλύτερες επιδόσεις και απεικονίσεις της Java desktop με βάση την παρούσα βιβλιοθήκη όπου είναι δυνατόν. Το .NET γίνεται όλο και περισσότερο κοινό σε open source και free software συστήματα λόγω του περιβάλλοντος GNOME και χρησιμοποιώντας το Mono framework.

2.3.2 Εφαρμογές server

Αυτή είναι ίσως η αρένα στην οποία οι δυο πλατφόρμες είναι πολύ πιθανό να χαρακτηριστούν ως αντίπαλοι. Η Java μέσω της Java EE (γνωστή και ως Java Platform Enterprise Edition) πλατφόρμας και το .NET μέσω του ASP.NET, ανταγωνίζονται για τη δημιουργία web-based δυναμικού περιεχομένου και εφαρμογών.

Και οι δυο πλατφόρμες χρησιμοποιούνται και υποστηρίζονται σε αυτήν την αγορά με μια ποικιλία εργαλείων και υποστήριξη των προϊόντων που διατίθενται για τη Java EE και την .NET. Και οι δυο έχουν υψηλού κύρους υποστηρικτές. Για παράδειγμα στη Java, η Oracle περιλαμβάνει άμεση υποστήριξη για τη Java στις βάσεις δεδομένων της, ενώ η Google έχει χρησιμοποιήσει τη Java σε δυναμικά εργαλεία όπως το Gmail.

Μερικές από τις τρέχουσες συμφωνίες παραχώρησης αδειών εκμετάλλευσης της Sun για τη Java EE καθορίζουν τις πτυχές της πλατφόρμας Java ως εμπορικό μυστικό και απαγορεύουν στο τελικό χρήστη να συμβάλλει στη διάδοση του περιβάλλοντος Java σε τρίτους. Συγκεκριμένα, τουλάχιστον μια τρέχουσα άδεια για ένα πακέτο ανάπτυξης της Sun Java EE περιλαμβάνει τους ακόλουθους όρους: «Μπορείτε να κάνετε ένα απλό αντίγραφο ασφαλείας του λογισμικού αλλά δεν μπορείτε να αντιγράψετε, να τροποποιήσετε η να διανέμετε το λογισμικό-«Δεν μπορείτε να δημοσιεύσετε η να παρέχετε αποτελέσματα οποιονδήποτε δοκιμών στο λογισμικό σε τρίτους χωρίς την προηγούμενη γραπτή συγκατάθεση της Sun»-«Το λογισμικό είναι εμπιστευτικό και υπόκεινται σε πνευματικά δικαιώματα». Ωστόσο, ενώ το λογισμικό της Sun υπόκειται στους παραπάνω όρους, το API της Sun’s Java EE έχει υλοποιηθεί υπό την άδεια ανοικτής πηγής από τα project JBoss και JOnAS.

Η εφαρμογή ASP.NET της Microsoft δεν αποτελεί μέρος της τυποποιημένης CLI και τα επίσημα εργαλεία της Microsoft δεν είναι open source ή ελεύθερου λογισμικού και απαιτούν Windows servers. Ωστόσο ένα cross-platform ελεύθερου λογισμικού ASP.NET 2.0 αποτελεί μέρος του Mono project (εκτός από webparts και Web Services Enhancements).

2.4 Ενσωματωμένες εφαρμογές

2.4.1 Εφαρμογές mobile

Η Java έχει μια πολύ μεγάλη βάση στην αγορά κινητών τηλεφώνων και PDA με τις φθηνότερες συσκευές. Το λογισμικό της Java , συμπεριλαμβανομένων πολλών παιχνιδιών είναι σύνηθες.

Ενώ σχεδόν κάθε κινητό τηλέφωνο περιλαμβάνει μια JVM, τα χαρακτηριστικά αυτά δε χρησιμοποιούνται πάντα από τους χρήστες (κυρίως στη Βόρεια Αμερική). Αρχικά, οι εφαρμογές Java στα περισσότερα τηλέφωνα αποτελούνταν από συστήματα μενού, μικρά παιχνίδια, τα συστήματα για κατέβασμα ringtones κλπ. Ωστόσο, όλο και πιο ισχυρά κινητά πωλούνταν με απλές εφαρμογές προ-φορτωμένες όπως για παράδειγμα, λεξικά, ρολόι κόσμου(φωτεινότητα, ζώνες ώρας κλπ) και αριθμομηχανές. Μερικές από αυτές είναι γραμμένες σε Java αν και η χρήση τους από τους κατόχους των κινητών είναι ελάχιστη.

Τον Ιανουάριο του 2007 ο Steve Jobs πρότεινε το iPhone της Apple να μην υποστηρίζει την Java. Αξίζει να σημειωθεί ότι εκείνο τον καιρό η πλατφόρμα θεωρούνταν πανταχού παρούσα στην αγορά των κινητών τηλεφώνων και χρησιμοποιούταν συχνά για εφαρμογές κινητών τηλεφώνων. Αρκετοί διαφώνησαν με τη θέση του Jobs αλλά όταν το iPhone βγήκε τελικά στην αγορά είχε και Java αλλά και Adobe flash τεχνολογία που ευνοούσαν απλές web εφαρμογές με τη χρήση του Safari browser.

Το Μάιο του 2007 η Sun στο συνέδριο JavaOne ανακοίνωσε το JavaFX mobile ως άμεση απάντηση στην απόπειρα της Adobe να παρουσιάσει το Flash πάνω σε κινητές συσκευές.

Τον Οκτώβριο του 2007 η Apple υπέκυψε σε πιέσεις και ανακοίνωσε ότι από τις αρχές του 2008 το iPhone θα επιτρέπει την ανάπτυξη και άλλου λογισμικού εκτός από το Safari browser. Ούτε το Java αλλά ούτε και το Flash υποστηρίζονται σε αυτό το πλάνο και η Apple δεν επέτρεψε διερμηνεία σε καμιά γλώσσα να λειτουργεί στο κινητό τηλέφωνο.

2.4.2 Τεχνολογίες Home entertainment

Η Java έχει βρει μια αγορά στην ψηφιακή τηλεόραση όπου μπορεί να χρησιμοποιηθεί για να παρέχει λογισμικό το οποίο τοποθετείται παράλληλα με τον προγραμματισμό ή επεκτείνει τις δυνατότητες ενός συγκεκριμένου Set Top Box. Για παράδειγμα το TiVo έχει μια δυνατότητα που ονομάζεται “Home Media Engine” και η οποία επιτρέπει στο λογισμικό JavaTV να μεταδίδει ένα ολοκληρωμένο πρόγραμμα η να παρέχει επιπλέον δυνατότητες όπως για παράδειγμα προγράμματα επιχειρηματικών ειδήσεων.

Μια παραλλαγή της Java έχει γίνει δεκτή ως επίσημο εργαλείο λογισμικού για χρήση στην γενιά οπτικών δίσκων τεχνολογίας Blu-ray, μέσω της BD-J διαδραστικής πλατφόρμας. Αυτό σημαίνει ότι το περιεχόμενο της διαδραστικότητας, όπως τα μενού, παιχνίδια, downloads κλπ σε όλους τους Blu-ray οπτικούς δίσκους θα δημιουργούνται με μια παραλλαγή της πλατφόρμας Java. Ο εξοπλισμός του Blu-ray βγήκε στην αγορά το 2006 χωρίς μεγάλη απήχηση. Ωστόσο με την έκδοση του Play station 3 στα τέλη του 2006 και αρχές του 2007 υπήρξε μια ώθηση στη συγκεκριμένη πλατφόρμα.

Αντί να χρησιμοποιεί Java, το HD DVD(ο διάδοχος του DVD) χρησιμοποιεί μια τεχνολογία που αναπτύχτηκε από κοινού από την Microsoft και την Disney, ονομάζεται iHD και βασίζεται στο XML, CSS, JavaScript και άλλες τεχνολογίες παρόμοιες με αυτές που χρησιμοποιούνται από τους τυπικούς web browsers.

Η BD-J πλατφόρμα API είναι πιο εκτεταμένη από τον αντίπαλο iHD, με περίπου 8000 μεθόδους και interfaces, σε αντίθεση με τις 400 του iHD. Και καθώς η Microsoft προωθεί το XML του iHD συμπεριλαμβάνοντάς το στα Windows Vista, το iHD εξακολουθεί να παραμένει νεοεισερχόμενο σε ένα τομέα της αγοράς όπου οι τεχνολογίες Java είναι ευρέως διαδεδομένες.

Ωστόσο, το γεγονός ότι το HD DVD format έχει εγκαταλειφτεί έναντι του Blu-ray, σημαίνει ότι το iHD δεν υποστηρίζεται πλέον από καμιά μορφή οπτικών δίσκων, καθιστώντας το BD-J format νικητή στη κατηγορία του.

2.5 Runtime inclusion στα λειτουργικά συστήματα

2.5.1 NET/Mono

Στα Windows, η Microsoft προωθεί το .NET ως την ναυαρχίδα πλατφόρμας ανάπτυξης, εντάσσοντας την στο runtime του .NET στα Windows XP Service Pack 2 και 3, στο Windows Server 2003, στα Windows Vista, στο Windows Server 2008 και στα Windows 7. Η Microsoft επίσης διανέμει το περιβάλλον προγραμματισμού Visual C# Express χωρίς κόστος.

Το .NET Framework 3.5 runtime δεν είναι προ-εγκατεστημένο στις εκδόσεις των Windows πριν από τα Vista SP1 και πρέπει να γίνει download από το χρήστη, κάτι που έχει επικριθεί λόγω του μεγάλου μεγέθους (65mb για να κατέβει το .NET 3.5).

Παρόλο που το .NET και το Mono δεν είναι εγκατεστημένα σε Mac-OS X, το Mono project μπορεί να γίνει download και να εγκατασταθεί δωρεάν για κάθε χρήστη Mac που θέλει να κατασκευάσει και να εκτελέσει την C# και το .NET software.

Από τις 13 Μαΐου του 2008, το Mono's system.Windows.Forms 2.0 είναι πλήρες από API(περιλαμβάνει το 100% κλάσεων, μεθόδων κλπ στο System.Windows.Forms 2.0 της Microsoft) . Επίσης το System.Windows.Forms 2.0 λειτουργεί πλήρες και στο Mac OS X.

Η C# και το CLI περιλαμβάνονται και χρησιμοποιούνται σε πολλά Linux και BSD λειτουργικά συστήματα μέσω του ελεύθερου λογισμικού Mono Project.

2.5.2 Java

Ξεκινώντας με τα XP SP1, τα Windows δεν χρησιμοποιούν το Java runtime περιβάλλον. Ωστόσο, σύμφωνα με ένα δελτίο τύπου το Σεπτέμβρη του 2003 συμφωνήθηκε να προ-εγκατασταθεί το JRE σε desktop και laptop μοντέλα. Αναφέρουμε την Acer, Dell, Gateway, Hewlett-Packard και την Toshiba. Οι εταιρείες αυτές αποτελούν στη πλειονότητα τους την επιλογή του καταναλωτή στις Ηνωμένες Πολιτείες της Αμερικής.

Η Java είναι προ-εγκατεστημένη σε όλους τους νέους υπολογιστές από το Mac OS X 10.0 και μετά. Επειδή η Apple υποστηρίζει το Java runtime για το Mac OS X, οι ενημερώσεις είναι συνήθως μια η δυο εκδόσεις πιο πίσω από τις εκδόσεις που είναι διαθέσιμες από τη Sun για άλλα λειτουργικά συστήματα και οι εκδόσεις της Java είναι συνήθως συνδεδεμένες με το συγκεκριμένο λειτουργικό σύστημα, οπότε οι νεώτερες εκδόσεις της Java δεν είναι συνήθως διαθέσιμες για τις παλαιότερες εκδόσεις του OS X.

Η Java έρχεται προ-εγκατεστημένη με πολλά εμπορικά κομμάτια του Unix, περιλαμβάνοντας αυτά της Sun, της IBM και της Hewlett Packard. Από τον Ιούνιο του 2009, το Fedora 9, Ubuntu 8.04 Debian, Slackware extra, Mandriva και OpenSUSE distributions είναι διαθέσιμα με το OpenJDK, που βασίζεται εντελώς σε ελεύθερο και ανοικτό κώδικα. Από τον Ιούνιο του 2008, το OpenJDK πέρασε όλες τις δοκιμές συμβατότητας στο Java SE 6 JCK και μπορεί να ισχυρίζεται ότι είναι πλήρως συμβατή με την εφαρμογή Java 6. Το OpenJDK μπορεί να τρέξει περίπλοκες εφαρμογές όπως το Netbeans, το Eclipse, το GlassFish, ή το JBoss.

Το λειτουργικό σύστημα Distributor License for Java (DLJ) είναι μια πρωτοβουλία της Sun για να διευκολύνει τα θέματα διανομής με τα λειτουργικά συστήματα που βασίζονται στο OpenSolaris η στο Linux.

Εάν η Java δεν είναι εγκατεστημένη σε έναν υπολογιστή από προεπιλογή, μπορεί να τη κατεβάσει ο χρήστης ως web plugin. Η διαδικασία web plugin έχει επικριθεί λόγω του μεγέθους του Java plugin. Σε αντίθεση με άλλα plugins, η Java download είναι ένα πλήρες runtime environment, που μπορεί να τρέχει όχι μόνο applets αλλά ολόκληρες εφαρμογές και δυναμικά WebStart apps. Γι αυτό το λόγο το μέγεθος του download είναι μεγαλύτερο από κάποια web plugins. Ωστόσο, σε σύγκριση με τη Java, άλλοι δημοφιλείς browser plugins έχουν μεγαλύτερα μεγέθη. Η Java 6 JRE είναι 13 MB, αλλά το Acrobat Reader είναι 33 MB, το QuickTime είναι 19 MB, το Windows Media Player 25 MB, το .NET Framework 3.0 runtime είναι 54 MB και το .NET Framework 3.5 runtime είναι 197 MB.

Στην εκδήλωση JavaOne τον Μάιο του 2007, η Sun ανακοίνωσε ότι τα ζητήματα ανάπτυξης με τη Java θα λύνονταν σε δυο σημαντικές ενημερώσεις στην Java 6 (οι αλλαγές δε θα αφορούσαν την νέα έκδοση Java 7). Αυτές περιλαμβάνουν:

- Τη παρουσίαση στον καταναλωτή μια νέας έκδοσης JRE με ένα μέγεθος περίπου 2 mb και τη δυνατότητα να κατεβάσουν τα υπόλοιπα 9 mb σε τμήματα χρησιμοποιώντας τη μεθοδολογία on-demand.
- Την ανάπτυξη μιας drop-in πολλαπλής πλατφόρμας JavaScrict κώδικα, η οποία μπορεί να χρησιμοποιηθεί από μια ιστοσελίδα για να εγκατασταθεί το απαραίτητο JRE για κάποιο applet η για να τρέξει κάποιο Rich Internet Application, εάν είναι απαραίτητο.
- Βελτίωση στην υποστήριξη για αυτόματο download ενημερώσεων για το JRE.
- Υποστήριξη pre-loading για το JRE , ώστε τα applets και οι εφαρμογές που είναι γραμμένες σε Java να ξεκινούν σχεδόν ακαριαία.

Κεφάλαιο 3

Γλώσσα προγραμματισμού C#

3.1 Εισαγωγή στη γλώσσα προγραμματισμού C#

Η C# είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού που είναι προσιτή προς τον χρήστη και παρέχει ασφάλεια τύπων. Δίνει τη δυνατότητα στους προγραμματιστές να αναπτύξουν μια ποικιλία ασφαλών και σταθερών εφαρμογών που να μπορούν να εκτελεστούν στο .NET Framework. Με την χρήση της γλώσσας C# μπορούν να δημιουργηθούν παραδοσιακές εφαρμογές Windows client, XML Web Services, κατανεμημένα συστατικά, εφαρμογές client-server, εφαρμογές βάσεων δεδομένων και πολλά άλλα. Η Visual C# 2010 παρέχει ένα βελτιωμένο επεξεργαστή κώδικα, σχεδιαστές διεπαφών χρήστη που είναι εύκολοι στην χρήση, ενοποιημένους διορθωτές καθώς και άλλα εργαλεία τα οποία διευκολύνουν την ανάπτυξη εφαρμογών που είναι βασισμένες στην έκδοση 4.0 της γλώσσας C# και στην έκδοση 4.0 του .NET Framework.

Η σύνταξη της γλώσσας C# είναι highly expressive, που σημαίνει ότι δίνει τη δυνατότητα να γραφτεί ο ίδιος κώδικας με λιγότερες γραμμές εντολών. Οι λιγότερες γραμμές εντολών σημαίνουν ότι είναι πιο εύκολο να γραφτεί και να επεκταθεί ο κώδικας, καθώς και να διατηρηθεί.. Επιπλέον η γλώσσα C# είναι απλή και εύκολη στην εκμάθηση. Η curly-brace σύνταξη της C# είναι εύκολα αναγνωρίσιμη από οποιονδήποτε χρήστη που είναι εξοικειωμένος με τις γλώσσες C, C++ και Java. Οι προγραμματιστές που γνωρίζουν οποιαδήποτε από τις παραπάνω γλώσσες έχουν την ικανότητα, έστω και τυπικά, να ξεκινήσουν να δουλεύουν παραγωγικά με την C# σε πολύ σύντομο χρονικό διάστημα. Η σύνταξη της C# απλοποιεί πολλές από τις πολυπλοκότητες της γλώσσας C++ και παρέχει κάποια ισχυρά(powerful) χαρακτηριστικά, όπως οι nullable value types, οι απαριθμήσεις, τα delegates, οι

εκφράσεις λάμδα (lambda expressions) και η άμεση πρόσβαση στην μνήμη, χαρακτηριστικά τα οποία δεν υπάρχουν στην Java. Η C# υποστηρίζει generic μεθόδους και τύπους που παρέχουν αυξημένη ασφάλεια τύπων και αποδοτικότητα, καθώς και iterators τα οποία είναι αντικείμενα που επιτρέπουν στον προγραμματιστή να διασχίσει τα στοιχεία μιας συλλογής.

Η γλώσσα C# ως αντικειμενοστραφής, υποστηρίζει τις έννοιες της ενθυλάκωσης, κληρονομικότητας και πολυμορφισμού. Όλες οι μεταβλητές και οι μέθοδοι, και η κύρια μέθοδος, είναι ενθυλακωμένες μέσα στον ορισμό των κλάσεων.

Η C# δίνει τη δυνατότητα ανάπτυξης συστατικών λογισμικού μέσω κάποιων καινοτόμων δομών γλώσσας, όπως:

- Τα delegates είναι μια ενθυλακωμένη μέθοδος υπογραφών που επιτρέπουν τις κοινοποιήσεις γεγονότων, ασφαλούς τύπου.
- Ιδιότητες που χρησιμεύουν ως accessors για τις μεταβλητές των private members.
- Γνωρίσματα που παρέχουν δηλωτικά μεταδεδομένα για τους τύπους στο χρόνο εκτέλεσης
- Inline σχόλια εγγράφων (documentation comments) HTML.
- Παροχή ενσωματωμένων ικανοτήτων αναζήτησης σε μια ποικιλία πηγών δεδομένων, μέσω της LINQ

3.2 Γενικά στοιχεία της C#

3.2.1 Τύποι δεδομένων και λέξεις κλειδιά

Ένας τύπος δεδομένων μπορεί να περιγραφεί είτε ως:

- Ενσωματωμένος τύπος δεδομένων, όπως είναι ο int και ο char, ή
- Τύπος δεδομένων καθορισμένος από τον χρήστη, όπως είναι οι κλάσεις και οι διεπαφές.

Ένας άλλος τρόπος διαχωρισμού των τύπων δεδομένων είναι σε reference type και value type.

Προτού αποθηκευτεί μια τιμή σε μια μεταβλητή, ο τύπος της μεταβλητής πρέπει να προσδιοριστεί όπως στο παρακάτω παράδειγμα:

```
int a = 1;
string s = "Hello";
XmlDocument tempDocument = new XmlDocument();
```

Οι τύποι πρέπει να προσδιορίζονται και για τους απλούς και για τους ενσωματωμένους τύπους ως int, και για τους απλούς ή τροποποιημένους τύπους ως XmlDocument.

Η C# περιλαμβάνει υποστήριξη για τους παρακάτω ενοποιημένους τύπους:

Τύποι Δεδομένων	Range
Byte	0 .. 255
Sbyte	-128 .. 127
Short	-32,768 .. 32,767
Ushort	0 .. 65,535
Int	-2,147,483,648 .. 2,147,483,647
Uint	0 .. 4,294,967,295
Long	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807
Ulong	0 .. 18,446,744,073,709,551,615
Float	-3.402823e38 .. 3.402823e38
Double	-1.79769313486232e308 .. 1.79769313486232e308
Decimal	-79228162514264337593543950335 .. 79228162514264337593543950335
Char	A Unicode character.
String	A string of Unicode characters
Bool	True or False.
Object	An object.

Αυτές οι ονομασίες τύπων είναι ψευδώνυμα για τους προκαθορισμένους τύπους στο System namespace. Όλοι οι παραπάνω τύποι, εκτός των object και string, είναι value types.

Χρήση των ενσωματωμένων τύπων δεδομένων

Οι ενσωματωμένοι τύποι χρησιμοποιούνται μέσα σε ένα πρόγραμμα της γλώσσας C# με διάφορους τρόπους:

1. Ως μεταβλητές:

```
int answer = 42;
string greeting = "Hello, World!";
```

2. Ως σταθερές:

```
const int speedLimit = 55;
const double pi = 3.14159265358979323846264338327950;
```

3. Ως τιμές επιστροφής(return values) και ως παράμετροι:

```
long CalculateSum(int a, int b)
{
   long result = a + b;
   return result;
}
```

Μετατρέποντας τους τύπους δεδομένων

Κατά τη διάρκεια της μετατροπής των τύπων, η οποία γίνεται αυτόματα από τον μεταγλωττιστή, ή με την χρήση ενός cast, κατά την οποία ο προγραμματιστής εξαναγκάζει την μετατροπή και αναλαμβάνει τον κίνδυνο να χαθούν οι πληροφορίες.

Για παράδειγμα:

```
int i = 0;
double d = 0;

i = 10;
d = i;        // An implicit conversion

d = 3.5;
i = (int) d;  // An explicit conversion, or "cast"
```

Τύποι καθορισμένοι από τον χρήστη

Οποιοδήποτε από τα προγράμματα της γλώσσας C# περιέχει τύπους καθορισμένους από το χρήστη. Η τιμή του τύπου που είναι καθορισμένος από τον χρήστη ονομάζεται class. Για παράδειγμα:

```
    class Hello
//This program displays Hello World
{
   static public void Main()
   {
      System.Console.WriteLine("Hello World");
   }
}
```

Η τιμή του τύπου που είναι καθορισμένη από το χρήστη και μπορεί να χρησιμοποιηθεί αντί της αντίστοιχης class, ονομάζεται struct.
Για παράδειγμα:

```
    using System;
struct Test{
  static void Main()
   {
     System.Console.WriteLine("Hi there");
   }
}
```

Value types και Reference types

Όλοι οι διαχειριζόμενοι κώδικες υποστηρίζουν δύο είδη τύπων: τους reference types και value types. Η διαφορά βρίσκεται στην μνήμη, εκεί που αποθηκεύονται τα δεδομένα των τύπων.

Οι έννοιες των reference types και value types περιγράφονται καλύτερα με παράδειγμα. Ας υποθέσουμε ότι έχουμε τους τύπους string και int32. Ο τύπος string είναι reference type και ο int32 value type. Και οι δύο τύποι είναι primitive types, μπορούν να κληθούν μέθοδοι και στους δυο τύπους και εξάγονται και οι δύο από αντικείμενα.

Διαφέρουν όμως στον τρόπο με τον οποίο διαχειρίζονται οι μεταβλητές τους. Μια μεταβλητή string θα είναι πάντα reference σε κάποιο string αντικείμενο στην μνήμη ή θα υπάρχει μια μηδενική reference στην μνήμη(δείχνοντας ότι δεν κάνει reference κανένα αντικείμενο). Ενώ μια μεταβλητή int32 είναι η τιμή ενός ακεραίου και δεν κάνει τίποτα reference στην μνήμη. Συνεπώς μπορούμε να βάλουμε την τιμή null σε μια string reference variable, αλλά δεν μπορούμε να βάλουμε την τιμή null σε μια Int32 value variable.

Ο μόνος τρόπος για να διαπιστώσουμε πότε ένας τύπος είναι reference ή value type είναι να δούμε αν είναι καταχωρημένος ως class ή struct. Στην πρώτη περίπτωση έχουμε reference type ενώ στη δεύτερη value type.

Λέξεις κλειδιά

Οι λέξεις κλειδιά είναι προκαθορισμένα, κρατημένα(reserved) αναγνωριστικά, που έχουν ιδιαίτερη σημασία για τον μεταγλωττιστή. Δεν μπορούν να χρησιμοποιηθούν ως αναγνωριστικά στο πρόγραμμα εκτός και αν περιέχουν το @ σαν πρόθεμα. Για παράδειγμα, το @if είναι ένα ισχύον αναγνωριστικό ενώ το if δεν είναι επειδή με αυτή την μορφή είναι λέξη κλειδί.

Ο πρώτος πίνακας περιέχει λέξεις κλειδιά που είναι κρατημένα αναγνωριστικά σε οποιοδήποτε σημείο του προγράμματος C#.

Abstract	As	Base	Bool
Break	Byte	Case	Catch
Char	Checked	Class	Const
Continue	Decimal	Default	Delegate
Do	Double	Else	Enum
Event	Explicit	Extern	False
Finally	Fixed	Float	For
Foreach	Goto	If	Implicit
In	In(generic modifier)	Int	Interface
Internal	Is	Lock	Long
Namespace	New	Null	Object
Operator	Out	Out (generic modifier)	Override
Params	Private	Protected	Public
Readonly	Ref	Return	Sbyte

Sealed	Short	Sizeof	Stackalloc
Static	String	Struct	Switch
This	Throw	True	Try
Typeof	Uint	Ulong	Unchecked
Unsafe	Ushort	Using	Virtual
Void	Volatile	While	

3.2.2 Τελεστές και εκφράσεις

Η γλώσσα C# παρέχει ένα μεγάλο σύνολο τελεστών, οι οποίοι είναι σύμβολα που προσδιορίζουν ποιες λειτουργίες θα εκτελεστούν σε μια έκφραση. Οι λειτουργίες στους ενοποιημένους τύπους όπως = =, !=, <, >, <=, >=, binary +, binary -, ^, &, |, ~, ++, -- και sizeof γενικά επιτρέπονται στις enumerations. Επιπροσθέτως, πολλοί τελεστές μπορεί να υπερφορτωθούν από τον χρήστη και συνεπώς, να αλλάξει η ερμηνεία τους όταν εφαρμοστούν σε ένα τύπο που είναι καθορισμένος από τον χρήστη.

Ο ακόλουθος πίνακας περιέχει τους τελεστές της C# που είναι ομαδοποιημένοι κατά σειρά προτεραιότητας. Οι τελεστές μέσα στην ίδια ομάδα έχουν την ίδια σειρά προτεραιότητας.

Κατηγορία τελεστή	Τελεστές
Πρωταρχικοί	X,Y F(x) A[x] X++

	X - - New Typeof Checked Unchecked Default(T) Delegate ->
Μοναδιαίοι	+ - ! ~ ++x - - x T(x) True False & Sizeof
Multiplicative	* / %

Additive	+ -
Shift	<<, >>
Relational and type testing	< > <= >= Is As
Ισότητας	= = =!
Λογικό ΚΑΙ	&
Λογικό XOR	^
Λογικό Η	|
Conditional ΚΑΙ	&&
Conditional Η	||
Null-coalescing	??
Conditional	?:
Assignment and lambda expression	= += -=

	*= /= %= &= |= ^= <<= >>= => τελεστής lambda

Μια έκφραση είναι μια σειρά από ένα ή περισσότερα operands και από κανένα ή περισσότερους τελεστές τα οποία μπορούν να εκτιμηθούν σε μια απλή value,σε ένα αντικείμενο, σε μια μέθοδο ή σε ένα namespace. Οι εκφράσεις μπορεί να αποτελούνται από ένα literal value, από μια επίκληση μεθόδου, από ένα τελεστή και τα operands ή από ένα απλό όνομα. Τα απλά ονόματα μπορεί να είναι το όνομα μιας μεταβλητής, ενός τύπου μέλους, μιας παραμέτρου μιας μεθόδου, ενός namespace ή ενός τύπου.

Οι εκφράσεις μπορούν να χρησιμοποιήσουν τελεστές που με την σειρά τους χρησιμοποιούν άλλες εκφράσεις ως παραμέτρους ή κλήσεις μεθόδων των οποίων οι παράμετροι είναι με την σειρά τους άλλες κλήσεις μεθόδων. Έτσι οι εκφράσεις κυμαίνονται από απλές έως πολύ σύνθετες. Παρακάτω δίνονται δύο παραδείγματα εκφράσεων:

```
((x < 10) && ( x > 5)) || ((x > 20) && (x < 25))
System.Convert.ToInt32("35")
```

Τιμές εκφράσεων

Στα περισσότερα πλαίσια που χρησιμοποιούνται εκφράσεις, όπως σε δηλώσεις ή σε παραμέτρους μεθόδων, η έκφραση αναμένεται να εκτιμηθεί με κάποια τιμή. Εάν το x και το y είναι ακέραιοι, η έκφραση x+y εκτιμά μια αριθμητική τιμή. Η έκφραση new MyClass () εκτιμά μια reference σε μια νέα instance ενός αντικειμένου MyClass. Η έκφραση MyClass.ToString() αξιολογεί μια string επειδή αυτή είναι ο τύπος επιστροφής της μεθόδου. Παρόλο που η ονομασία ενός namespace κατατάσσεται σαν μια έκφραση, δεν καταχωρείται σε τιμή και συνεπώς δεν μπορεί να είναι ποτέ το τελικό αποτέλεσμα μιας έκφρασης. Ένα όνομα namespace δε γίνεται να περαστεί σε μια μέθοδο μιας παραμέτρου ή να χρησιμοποιηθεί σε μια νέα έκφραση ή να εκχωρηθεί σε μια μεταβλητή. Μπορεί μόνο να χρησιμοποιηθεί σαν υποσύνολο μιας μεγαλύτερης έκφρασης. Το ίδιο ισχύει για τους τύπους, για τα ονόματα ομάδων μεθόδων, καθώς και για το γεγονός(event) προσθήκης και αφαίρεσης accessors.

Κάθε τιμή έχει ένα συσχετισμένο τύπο. Για παράδειγμα, εάν το x και το y είναι μεταβλητές ακεραίου τύπου, η τιμή της έκφρασης x+y γράφεται επίσης ως int. Εάν η τιμή αντιστοιχηθεί σε μια μεταβλητή ενός διαφορετικού τύπου ή αν τα x και y είναι διαφορετικοί τύποι, εφαρμόζονται οι κανόνες μετατροπής των τύπων.

Literals και απλά ονόματα

Οι δύο πιο απλοί τύποι των εκφράσεων είναι τα απλά και τα literal ονόματα. Το literal είναι μια σταθερή τιμή η οποία δεν έχει όνομα. Στο παράδειγμα που ακολουθεί παρακάτω το 5 και το “hello world” είναι literal τιμές:

```
// Expression statements.
int i = 5;
string s = "Hello World";
```

Στο ίδιο παράδειγμα και το i και το s είναι απλά ονόματα που αναγνωρίζουν τοπικές μεταβλητές. Όταν οι μεταβλητές αυτές χρησιμοποιούνται σε μια έκφραση, το όνομα της μεταβλητής αποδίδεται στην τιμή που είναι προσωρινά αποθηκευμένη στην τοποθεσία της μεταβλητής στην μνήμη. Αυτό φαίνεται στο παρακάτω παράδειγμα:

```
int num = 5;
System.Console.WriteLine(num); // Output: 5
num = 6;
System.Console.WriteLine(num); // Output: 6
```

Εκφράσεις επίκλησης

Στο επόμενο παράδειγμα η κλήση dowork είναι μια έκφραση επίκλησης:

```
DoWork();
```

Μια μέθοδος επίκλησης απαιτεί το όνομα της μεθόδου, είτε ως όνομα όπως στο προηγούμενο παράδειγμα ή ως αποτέλεσμα μια άλλης έκφρασης, που ακολουθείται από παρένθεση και οποιαδήποτε παράμετρο μεθόδου. Μια delegate επίκληση χρησιμοποιεί το όνομα του delegate και τις παραμέτρους της μεθόδου σε παρένθεση. Οι επικλήσεις μεθόδων και delegate εκτιμούν την επιστροφή της τιμής της μεθόδου, εάν η μέθοδος επιστρέψει κάποια τιμή. Η μέθοδος που επιστρέφει κενή τιμή, δεν μπορεί να την χρησιμοποιήσει στη θέση κάποιας τιμής μέσα στην έκφραση.

Εκφράσεις lambda

Οι lambda εκφράσεις αντιπροσωπεύουν τις inline μεθόδους, οι οποίες δεν έχουν όνομα αλλά μπορούν να έχουν παραμέτρους εισαγωγής και πολλαπλές δηλώσεις. Χρησιμοποιούνται εκτενώς στη LINQ για να περάσει arguments σε μεθόδους. Οι εκφράσεις lambda μεταγλωττίζονται είτε σε delegates ή σε Δένδρα έκφρασης ανάλογα με το πλαίσιο στο οποίο χρησιμοποιούνται.

Δένδρα έκφρασης

Τα δένδρα έκφρασης επιτρέπουν στις εκφράσεις να αναπαρίστανται ως δομές δεδομένων. Χρησιμοποιούνται εκτενώς από τους παροχείς για να μεταφράζουν query εκφράσεις σε κώδικα που έχει νόημα σε κάποιο άλλο πλαίσιο, όπως η βάση δεδομένων SQL.

3.2.3 Δηλώσεις

Οι ενέργειες που κάνει ένα πρόγραμμα εκφράζονται σε δηλώσεις. Συνήθεις ενέργειες περιλαμβάνουν: τον καθορισμό των μεταβλητών, την αντιστοίχιση των τιμών, τους βρόχους συλλογών και τη διακλάδωση(branching) από ένα τμήμα του κώδικα σε ένα άλλο, ανάλογα με την υπάρχουσα συνθήκη. Η σειρά με την οποία εκτελούνται οι δηλώσεις ονομάζεται έλεγχος ροής ή ροή εκτέλεσης. Ο έλεγχος ροής μπορεί να ποικίλει κάθε φορά που εκτελείται το πρόγραμμα, ανάλογα με το πώς αυτό αντιδρά κατά την εισαγωγή των δεδομένων που γίνεται στο χρόνο εκτέλεσης.

Μια δήλωση μπορεί να αποτελείται από μόνο μια γραμμή κώδικα που στο τέλος της έχει ερωτηματικό ή από μια σειρά δηλώσεων μιας γραμμής σε ένα block. Ένα block δήλωσης περικλείεται από άγκιστρα {} και μπορεί να περιέχει εμφωλευμένα blocks. Ο παρακάτω κώδικας δείχνει δύο παραδείγματα:ένα παράδειγμα δήλωσης μιας γραμμής και ένα παράδειγμα block πολλαπλών γραμμών:

```
static void Main()
   {
      // Declaration statement.
      int counter;

      // Assignment statement.
      counter = 1;

      // Error! This is an expression, not an expression statement.
      // counter + 1;

      // Declaration statements with initializers are functionally
      // equivalent to pointA declaration statement followed by assignment statement:
```

```
        int[] radii = { 15, 32, 108, 74, 9 }; // Declare and initialize an array.
        const double pi = 3.14159; // Declare and initialize pointA constant.

        // foreach statement block that contains multiple statements.
        foreach (int radius in radii)
        {
            // Declaration statement with initializer.
            double circumference = pi * (2 * radius);

            // Expression statement (method invocation). A single-line
            // statement can span multiple text lines because line breaks
            // are treated as white space, which is ignored by the compiler.
            System.Console.WriteLine("Radius of circle #{0} is {1}. Circumference =
{2:N2}",
                                     counter, radius, circumference);

            // Expression statement (postfix increment).
            counter++;

        } // End of foreach statement block
    } // End of Main method body.
} // End of SimpleStatements class.
/*
   Output:
    Radius of circle #1 = 15. Circumference = 94.25
    Radius of circle #2 = 32. Circumference = 201.06
    Radius of circle #3 = 108. Circumference = 678.58
    Radius of circle #4 = 74. Circumference = 464.96
    Radius of circle #5 = 9. Circumference = 56.55
*/
```

Τύποι δηλώσεων

Ο παρακάτω πίνακας περιέχει τους διάφορους τύπους δηλώσεων σε γλώσσα C# και τις συσχετισμένες λέξεις κλειδιά:

Κατηγορία	Λέξεις κλειδιά C#
Δηλώσεις διασάφηση(declaration)	Μια δήλωση διασάφησης εισάγει μια νέα μεταβλητή ή σταθερά. Μια διασάφηση μεταβλητής μπορεί προαιρετικά να βάλει μια τιμή στην μεταβλητή. Στη διασάφηση σταθεράς η εισαγωγή

	τιμής απαιτείται
	// Variable declaration statements.
	double area;
	double radius = 2;
	
	// Constant declaration statement.
	const double pi = 3.14159;

Δηλώσεις εκφράσεων	Οι δηλώσεις εκφράσεων που έχουν τη δυνατότητα να υπολογίζουν μια τιμή θα πρέπει να την αποθηκεύουν σε μια μεταβλητή.
	// Expression statement (assignment).
	area = 3.14 * (radius * radius);
	
	// Error. Not pointA statement because no assignment:
	//circ * 2;
	
	// Expression statement (method invocation).
	System.Console.WriteLine();
	
	// Expression statement (new object creation).
	System.Collections.Generic.List<string> strings =
	new System.Collections.Generic.List<string>();

Δηλώσεις επιλογής	Οι δηλώσεις επιλογής δίνουν τη δυνατότητα μετάβασης σε διαφορετικά τμήματα του κώδικα, ανάλογα με τον αριθμό των συνθηκών. Οι μορφές των συνθηκών είναι: if, else, switch, case
Δηλώσεις iteration	Οι δηλώσεις iteration δίνουν τη δυνατότητα να κάνουμε loop σε συλλογές όπως οι μήτρες ή να εκτελούνται τα ίδια σύνολα δηλώσεων επαναλαμβανόμενα μέχρι να ικανοποιηθεί μια συγκεκριμένη συνθήκη. Οι δηλώσει Iteration είναι: for, foreach, while, do, in
Δηλώσεις jump	Οι δηλώσεις jump μεταφέρουν τον έλεγχο από ένα τμήμα του κώδικα σε ένα άλλο. Τέτοιες δηλώσεις είναι: break, continue, default, goto, return, yield

Δηλώσεις χειρισμού εξαίρεσης	Οι δηλώσεις χειρισμού εξαίρεσης δίνουν τη δυνατότητα ανάκαμψης από συνθήκες εξαίρεσης που είναι πιθανό να προκύψουν κατά το χρόνο εκτέλεσης, π.χ.:throw, try-catch, try-finally, try-catch-finally
Ελεγμένες και μη ελεγμένες δηλώσεις	Οι ελεγμένες και μη ελεγμένες δηλώσεις δίνουν τη δυνατότητα να διευκρινιστεί εάν οι αριθμητικές λειτουργίες έχουν τη δυνατότητα να προκαλέσουν υπερχείλιση όταν το αποτέλεσμα αποθηκεύεται σε μια μεταβλητή που είναι πολύ μικρή για να κρατήσει την τιμή του αποτελέσματος.
Fixed δήλωση	Εμποδίζει τον συλλογέα απορριμμάτων από την μετεγκατάσταση μιας κινητής μεταβλητής
Δήλωση lock	Περιορίζει την πρόσβαση των blocks του κώδικα σε ένα νήμα την φορά
Δηλώσεις με επισήμανση	Μπορεί να δοθεί σε μια δήλωση μια επισήμανση και μετά να χρησιμοποιηθεί η λέξη κλειδί goto για να μεταβούμε στη δήλωση με την επισήμανση.

Κενή δήλωση	Η κενή δήλωση αποτελείται μόνο από ένα ερωτηματικό. Δεν κάνει κάποια ενέργεια και χρησιμεύει όπου απαιτείται δήλωση αλλά δεν πρέπει να πραγματοποιηθεί κάποια ενέργεια. Τα ακόλουθα παραδείγματα δείχνουν δύο χρήσεις για μια κενή δήλωση:
	`void ProcessMessages()`
	`{`
	`  while (ProcessMessage())`
	`  ; // Statement needed here.`
	`}`
	
	`void F()`
	`{`
	`  //...`
	`  if (done) goto exit;`
	`//...`
	`exit:`
	`  ; // Statement needed here.`

	}

Ενσωματωμένες δηλώσεις

Κάποιες δηλώσεις όπως οι do, while, for και foreach, έχουν πάντα μια ενσωματωμένη δήλωση που τις ακολουθεί. Αυτή η δήλωση μπορεί να είναι είτε απλή ή πολλαπλές δηλώσεις περικλειόμενες από άγκιστρα {}σε ένα block δήλωσης. Ακόμα και οι ενσωματωμένες δηλώσεις μιας γραμμής μπορούν να περικλείονται από άγκιστρα όπως στο παράδειγμα:

```
// Recommended style. Embedded statement in pointA block.
foreach (string s in System.IO.Directory.GetDirectories(
                    System.Environment.CurrentDirectory))
{
   System.Console.WriteLine(s);
}

// Not recommended.
foreach (string s in System.IO.Directory.GetDirectories(
                    System.Environment.CurrentDirectory))
   System.Console.WriteLine(s);
```

Μια ενσωματωμένη δήλωση που δεν περικλείεται σε άγκιστρα δεν μπορεί να είναι δήλωση διασάφησης ή δήλωση με επισήμανση, όπως φαίνεται και παρακάτω:

```
if(pointB == true)
   //Error CS1023:
   int radius = 5;
```

Η ενσωματωμένη δήλωση πρέπει να τοποθετηθεί στο block για να διορθωθεί το σφάλμα:

```
if (b == true)
{
   // OK:
   System.DateTime d = System.DateTime.Now;
   System.Console.WriteLine(d.ToLongDateString());
}
```

Εμφωλευμένα μπλοκ δηλώσεων

Τα μπλοκ δηλώσεων μπορούν να είναι εμφωλευμένα όπως φαίνεται παρακάτω:

```
foreach (string s in System.IO.Directory.GetDirectories(
   System.Environment.CurrentDirectory))
{
   if (s.StartsWith("CSharp"))
   {
      if (s.EndsWith("TempFolder"))
      {
         return s;
      }
   }

}
return "Not found.";
```

Unreachable δηλώσεις

Εάν ο μεταγλωττιστής καθορίζει πως ο έλεγχος ροής δεν μπορεί να φθάσει μια συγκεκριμένη δήλωση υπό οποιασδήποτε συνθήκες, παράγει μια προειδοποίηση CS0162, όπως στο παράδειγμα:

```
// An over-simplified example of unreachable code.
const int val = 5;
if (val < 4)
{
   System.Console.WriteLine("I'll never write anything."); //CS0162
```

```
}
```

3.2.4 Μέθοδοι

Μια μέθοδος είναι ο κωδικός ενός μπλοκ που περιέχει μια σειρά από δηλώσεις. Το πρόγραμμα αναγκάζει τις δηλώσεις να εκτελεστούν καλώντας μια μέθοδο και προσδιορίζει τα όποια απαιτούμενα arguments της μεθόδου. Στη C#, κάθε εκτελέσιμη οδηγία αποδίδεται μέσα στο πλαίσια της μεθόδου. Η κύρια μέθοδος είναι το εισαγωγικό σημείο για κάθε εφαρμογή C# και καλείται από το common language runtime (CLR) όταν το πρόγραμμα εκκινείτε.

Οι παράμετροι των μεθόδων περικλείονται από παρενθέσεις και διαχωρίζονται από κόμματα. Οι κενές παρενθέσεις δείχνουν ότι δεν απαιτείται παράμετρος από την μέθοδο. Η παρακάτω κλάση περιέχει τρεις μεθόδους:

```
abstract class Motorcycle
{
   // Anyone can call this.
   public void StartEngine() {/* Method statements here */ }

   // Only derived classes can call this.
   protected void AddGas(int gallons) { /* Method statements here */ }

   // Derived classes can override the base class implementation.
   public virtual int Drive(int miles, int speed) { /* Method statements here */ return
1; }

   // Derived classes must implement this.
   public abstract double GetTopSpeed();
}
```

Οι μέθοδοι μπορούν να λάβουν οποιονδήποτε αριθμό παραμέτρων ή καμία παράμετρο. Οι μέθοδοι μπορεί να επιστρέψουν οποιονδήποτε τύπο ή την λέξη «κενός» για να δείξουν ότι δεν επιστρέφουν τύπο.

Υπάρχουν δύο είδη μεθόδων:οι στατικές και οι instance. Οι στατικές ορίζονται από την λέξη κλειδί static. Μπορούν να κληθούν χωρίς να δημιουργήσουν instance του ενσωματωμένου τύπου. Αυτή είναι η σύνταξη για την κλήση μιας στατικής μεθόδου:

```
TypeName.MethodName();
```

Το παρακάτω παράδειγμα δείχνει τον ορισμό και την χρήση μιας σειράς μεθόδων, οι οποίες εμφανίζονται με κόκκινη γραμματοσειρά.

```
    // The next three lines of code test the methods in SomeClass

    SomeClass.Method1();


    SomeClass obj = new SomeClass();
    Int32 z = obj.SumMethod(12, 13);
    Console.WriteLine(z);
  }
}
 class SomeClass{
  public static void Method1(){
    Console.WriteLine("Here I am inside of SomeClass.Method1");
  }


  public Int32 SumMethod(Int32 x, Int32 y){
    return x+y;
  }
}
```

Παρατηρούμε ότι το SomeClass ορίζει δυο μεθόδους που ονομάζονται Method1 και SumMethod.Method1. Η μέθοδος Method1 είναι στατική ενώ η μέθοδος SumMethod είναι μια μέθοδος instance. Ο κώδικας που καλεί την Method1 δεν απαιτεί κάποια instance της SomeClass, ενώ ο κώδικας που καλεί την SumMethod χρειάζεται μια νέα instance της SomeClass, προτού καλέσει την μέθοδο. Παρατηρούμε επίσης, ότι η SumMethod λαμβάνει δυο παραμέτρους και επιστρέφει τιμή, ενώ η Method1 δεν παίρνει καμία παράμετρο και δεν επιστρέφει καμία τιμή.

Παράμετροι vs arguments

Ο ορισμός της μεθόδου διευκρινίζει τα ονόματα και τους τύπους των παραμέτρων που απαιτούνται. Όταν ο κώδικας κλήσης καλεί την μέθοδο, παρέχει συγκεκριμένες τιμές για κάθε παράμετρο, που λέγονται arguments. Οι arguments πρεπει να είναι συμβατές με τον τύπο της παραμέτρου αλλά η ονομασία των arguments,εφόσον υπάρχει, που χρησιμοποιείται στον κώδικα κλήσης δε χρειάζεται να είναι η ίδια με την ονομασία της παραμέτρου που έχει καθοριστεί από την μέθοδο. Παραδείγματος χάριν:

```
public void Caller()
{
   int numA = 4;
   // Call with an int variable.
   int productA = Square(numA);

   int numB = 32;
   // Call with another int variable.
   int productB = Square(numB);

   // Call with an integer literal.
   int productC = Square(12);

   // Call with an expression that evaulates to int.
   productC = Square(productA * 3);
}

int Square(int i)
{
   // Store input argument in a local variable.
   int input = i;
   return input * input;
}
```

3.2.5 Strings

Το string είναι ένα αντικείμενο τύπου String του οποίου η τιμή εκφράζεται σε χαρακτήρες κειμένου.

Μπορούμε να δηλώσουμε και να ορίσουμε strings με διάφορους τρόπους, όπως στο παράδειγμα:

```
// Declare without initializing.
        string message1;

        // Initialize to null.
        string message2 = null;

        // Initialize as an empty string.
        // Use the Empty constant instead of the literal "".
        string message3 = System.String.Empty;

        //Initialize with a regular string literal.
        string oldPath = "c:\\Program Files\\Microsoft Visual Studio 8.0";

        // Initialize with a verbatim string literal.
        string newPath = @"c:\Program Files\Microsoft Visual Studio 9.0";

        // Use System.String if you prefer.
        System.String greeting = "Hello World!";

        // In local variables (i.e. within a method body)
        // you can use implicit typing.
        var temp = "I'm still a strongly-typed System.String!";

        // Use a const string to prevent 'message4' from
        // being used to store another string value.
        const string message4 = "You can't get rid of me!";

        // Use the String constructor only when creating
        // a string from a char*, char[], or sbyte*. See
        // System.String documentation for details.
        char[] letters = { 'A', 'B', 'C' };
        string alphabet = new string(letters);
```

Regular και Verbatim String Literals

Χρησιμοποιούμε string literals όταν πρέπει να ενσωματώσουμε escape characters, οι οποίοι παρέχονται από τη C#, όπως στο παράδειγμα:

```
string columns = "Column 1\tColumn 2\tColumn 3";
//Output: Column 1       Column 2       Column 3

string rows = "Row 1\r\nRow 2\r\nRow 3";
/* Output:
  Row 1
  Row 2
  Row 3
*/

string title = "\"The \u00C6olean Harp\", by Samuel Taylor Coleridge";
//Output: "The Æolean Harp", by Samuel Taylor Coleridge
```

Τα verbatim strings χρησιμοποιούνται για μεγαλύτερη ευκολία στην ανάγνωση του κειμένου όταν το κείμενο περιέχει blackslash χαρακτήρες, όπως για παράδειγμα στις διαδρομές των αρχείων (file paths). Επειδή διατηρούν new line χαρακτήρες σαν μέρος του κειμένου string, μπορούν να χρησιμοποιηθούν για να ορίσουν τα multiline strings. Το παρακάτω παράδειγμα δείχνει κάποιες κοινές χρήσεις για τα verbatim strings:

```
  string filePath = @"C:\Users\scoleridge\Documents\";
        //Output: C:\Users\scoleridge\Documents\

        string text = @"My pensive SARA ! thy soft cheek reclined
Thus on mine arm, most soothing sweet it is
To sit beside our Cot,...";
        /* Output:
        My pensive SARA ! thy soft cheek reclined
          Thus on mine arm, most soothing sweet it is
          To sit beside our Cot,...
        */
```

```
string quote = @"Her name was ""Sara.""";
//Output: Her name was "Sara."
```

Format strings

Ένα format string είναι αυτό, του οποίου τα περιεχόμενα μπορούν να καθοριστούν δυναμικά, κατά το χρόνο εκτέλεσης. Με την χρήση της στατικής μεθόδου Format δημιουργούμε ένα format string. Τα ενσωματωμένα placeholders που είναι ανάμεσα στα εισαγωγικά αντικαθίστανται από τιμές στο χρόνο εκτέλεσης.

```
class FormatString
{
   static void Main()
   {
      // Get user input.
      System.Console.WriteLine("Enter a number");
      string input = System.Console.ReadLine();

      // Convert the input string to an int.
      int j;
      System.Int32.TryParse(input, out j);

      // Write a different string each iteration.
      string s;
      for (int i = 0; i < 10; i++)
      {
         // A simple format string with no alignment formatting.
         s = System.String.Format("{0} times {1} = {2}", i, j, (i * j));
         System.Console.WriteLine(s);
      }

      //Keep the console window open in debug mode.
      System.Console.ReadKey();
   }
}
```

Μπορεί να χρησιμοποιηθεί ο συμβολισμός της μήτρας με δείκτη μια τιμή, για να αποκτηθεί πρόσβαση, μόνο για ανάγνωση, σε ανεξάρτητους χαρακτήρες όπως στο παράδειγμα:

```
string s5 = "Printing backwards";

for (int i = 0; i < s5.Length; i++)
{
   System.Console.Write(s5[s5.Length - i - 1]);
}
// Output: "sdrawkcab gnitnirP"
```

3.2.6 Πίνακες

Ένας πίνακας είναι μια δομή δεδομένων που περιέχει μια σειρά από μεταβλητές του ίδιου τύπου. Οι πίνακες καθορίζονται από ένα τύπο: type[] arrayName;

Ένας πίνακας έχει τις παρακάτω ιδιότητες:

- Μπορεί να είναι ενιαίας διάστασης, πολυδιάστατος ή jagged
- Η προκαθορισμένη τιμή ενός αριθμητικού στοιχείου του πίνακα μπορεί να είναι το μηδέν, και τα στοιχεία reference και αυτά ορισμένα στο μηδέν.
- Ο πίνακας jagged είναι ένας πίνακας που ανήκει σε άλλους πίνακες και ως εκ τούτου τα στοιχεία του και οι τύποι reference έχουν αρχική τιμή το μηδέν.
- Οι πίνακες περιέχουν το δείκτη μηδέν: ένας πίνακας με n στοιχεία ξεκινάει από το 0 έως το n-1.
- Τα στοιχεία του πίνακα μπορούν να είναι οποιουδήποτε τύπου, συμπεριλαμβανομένου και του τύπου πίνακα.
- Οι τύποι πίνακα είναι τύποι reference που εξάγονται από την αφαιρούμενη βάση τύπων Array. Από την στιγμή που ο συγκεκριμένος τύπος εφαρμόζει

τα IEnumerable, IEnumerable <T>, μπορεί να χρησιμοποιηθεί το iteration foreach σε όλες τους πίνακες στη C#.

Πίνακες ενιαίας διάστασης

Ορίζουμε ένα πίνακα πέντε ακεραίων όπως στο παράδειγμα:

```
int[] array = new int[5];
```

Ο πίνακας περιέχει στοιχεία από την array[0] έως την array[4]. Ο νέος τελεστής χρησιμοποιείται για να δημιουργήσει ένα πίνακα και να καθορίσει τα στοιχεία του πίνακα στις αρχικές τους τιμές. Σε αυτό το παράδειγμα τα στοιχεία ορίζονται στην τιμή μηδέν.

Ένας πίνακας που αποθηκεύει στοιχεία string ορίζεται με τον ίδιο τρόπο:

```
string[] stringArray = new string[6];
```

Αρχικοποίηση πινάκων

Είναι δυνατό να οριστεί ένας πίνακας κατά τη δήλωση. Σε τέτοια περίπτωση, δε χρειάζεται να διευκρινιστεί η κατάταξη επειδή ο πίνακας έχει ήδη προμηθευτεί τον αριθμό τον στοιχείων από την λίστα ορισμού: Για παράδειγμα:

```
int[] array1 = new int[] { 1, 3, 5, 7, 9 };
```

Ένας πίνακας με strings ορίζεται με τον ίδιο τρόπο. Στο παρακάτω παράδειγμα τα στοιχεία ορίζονται με βάση το όνομα της ημέρας:

```
string[] weekDays = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
```

Όταν ορίζουμε ένα πίνακα κατά τη δήλωση χρειαζόμαστε τις παρακάτω συντομεύσεις:

```
int[] array2 = { 1, 3, 5, 7, 9 };
string[] weekDays2 = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
```

Είναι δυνατό να δηλωθεί μια μεταβλητή πίνακα χωρίς να οριστεί, αλλά θα πρέπει να χρησιμοποιηθεί ο τελεστής new όταν αντιστοιχιστεί ο πίνακας στην μεταβλητή:

```
int[] array3;
array3 = new int[] { 1, 3, 5, 7, 9 };   // OK
//array3 = {1, 3, 5, 7, 9};   // Error
```

Το αποτέλεσμα της παρακάτω δήλωσης εξαρτάται από το αν το SomeType είναι value ή reference τύπος. Στην πρώτη περίπτωση, η δήλωση δημιουργεί ένα πίνακα με 10 instances του τύπου SomeType. Στη δεύτερη περίπτωση, η δήλωση δημιουργεί μια μήτρα 10 στοιχείων, καθένα από τα οποία ορίζεται με την μηδενική reference:

```
SomeType[] array4 = new SomeType[10];
```

Πολυδιάστατοι πίνακες

Οι πίνακες μπορεί να έχουν περισσότερες από μια διαστάσεις. Για παράδειγμα η παρακάτω δήλωση δημιουργεί πίνακες δυο διαστάσεων με 4 σειρές και 2 στήλες:

```
int[,] array = new int[4, 2];
```

Το ακόλουθο παράδειγμα δημιουργεί πίνακα με 3 διαστάσεις: 4,2, και 3

```
int[, ,] array1 = new int[4, 2, 3];
```

Αρχικοποίηση πινάκων

Μπορεί να οριστεί ένας πίνακας κατά τη δήλωση, όπως φαίνεται στο παράδειγμα:

```
// Two-dimensional array.
int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
// Same array with dimensions specified.
int[,] array2Da = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };

// Three-dimensional array.
int[, ,] array3D = new int[,,] { { { 1, 2, 3 }, { 4, 5, 6 } }, { { 7, 8, 9 }, { 10, 11, 12
} } };
// Same array with dimensions specified.
int[, ,] array3Da = new int[2, 2, 3] { { { 1, 2, 3 }, { 4, 5, 6 } }, { { 7, 8, 9 }, { 10,
11, 12 } } };


// Three-dimensional array.
int[, ,] array_3D = new int[,,] { { { 1, 2 }, { 3, 4 }, { 5, 6 } },
                                  { { 7, 8 }, { 9, 10 }, { 11, 12 } } };
// Same array with dimensions specified.
int[, ,] array_3Da = new int[2, 3, 2] { { { 1, 2 }, { 3, 4 }, { 5, 6 } },
                                        { { 7, 8 }, { 9, 10 }, { 11, 12 } } };
```

Μπορεί να οριστεί χωρίς να διευκρινιστεί η σειρά:

```
int[,] array4 = { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
```

Εάν επιλέξουμε να δηλώσουμε μια μεταβλητή πίνακα χωρίς να την έχουμε ορίσει, πρέπει να χρησιμοποιηθεί ο τελεστής new για να αντιστοιχηθεί ο πίνακας με την μεταβλητή:

```
int[,] array5;
array5 = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };   // OK
//array5 = {{1,2}, {3,4}, {5,6}, {7,8}};   // Error
```

Το επόμενο παράδειγμα αντιστοιχίζει μια τιμή σε ένα συγκεκριμένο στοιχείο του πίνακα:

```
array5[2, 1] = 25;
```

Παρομοίως, παίρνει την τιμή ενός συγκεκριμένου στοιχείου του πίνακα και το αντιστοιχεί με την μεταβλητή elementValue:

```
int elementValue = array5[2, 1];
```

Σε αυτό το παράδειγμα ορίζει τα στοιχεία του πίνακα στις αρχικές τους τιμές(εξαιρούνται οι πίνακες jagged):

```
int[,] array6 = new int[10, 10];
```

Πίνακες Jagged

Ο πίνακας jagged είναι ένας πίνακας του οποίου τα στοιχεία είναι πίνακες. Τα στοιχεία του μπορούν να είναι διαφορετικού μεγέθους και διαφορετικών διαστάσεων. Ένας τέτοιος πίνακας καλείται επίσης και «πίνακας των πινάκων». Τα παρακάτω παραδείγματα δείχνουν πως δηλώνονται, ορίζονται και πως γίνονται προσβάσιμες οι πίνακες jagged.

Το παρακάτω παράδειγμα είναι μια δήλωση ενός μονοδιάστατου πίνακα που έχει τρία στοιχεία, κάθε ένα από τα οποία είναι ένας μονοδιάστατος πίνακας ακεραίων:

```
int[][] jaggedArray = new int[3][];
```

Πριν χρησιμοποιηθεί ο jaggedArray πρέπει να οριστούν τα στοιχεία του, τα οποία ορίζονται ως εξής:

```
jaggedArray[0] = new int[5];
jaggedArray[1] = new int[4];
jaggedArray[2] = new int[2];
```

Κάθε ένα από τα στοιχεία είναι ένας μονοδιάστατος πίνακας ακεραίων. Το πρώτο στοιχείο είναι ένας πίνακας 5 ακεραίων, το δεύτερο 4 και το τρίτο 2.

Είναι δυνατό να χρησιμοποιήσουμε αρχικοποιήσεις για να αντιστοιχίσουμε τα στοιχεία των πινάκων με τιμές, στην περίπτωση που δεν χρειάζεται το μέγεθος του πίνακα:

```
jaggedArray[0] = new int[] { 1, 3, 5, 7, 9 };
jaggedArray[1] = new int[] { 0, 2, 4, 6 };
jaggedArray[2] = new int[] { 11, 22 };
```

Μπορούμε να ορίσουμε τους πίνακες κατά την δήλωση ως εξής:

```
int[][] jaggedArray2 = new int[][]
{
   new int[] {1,3,5,7,9},
   new int[] {0,2,4,6},
   new int[] {11,22}
};
```

Μπορούμε να έχουμε πρόσβαση σε ανεξάρτητα στοιχεία πινάκων όπως στο παράδειγμα:

```
// Assign 77 to the second element ([1]) of the first array ([0]):
jaggedArray3[0][1] = 77;

// Assign 88 to the second element ([1]) of the third array ([2]):
jaggedArray3[2][1] = 88;
```

Είναι δυνατό να συνδυάσουμε τους πολυδιάστατους πίνακες με τους jagged. Το παρακάτω παράδειγμα είναι μια δήλωση και μια αρχικοποίηση ενός μονοδιάστατου jagged πίνακα που περιέχει τρία δισδιάστατα στοιχεία πίνακα, διαφορετικού μεγέθους:

```
int[][,] jaggedArray4 = new int[3][,]
{
   new int[,] { {1,3}, {5,7} },
   new int[,] { {0,2}, {4,6}, {8,10} },
   new int[,] { {11,22}, {99,88}, {0,9} }
};
```

Μπορούμε να έχουμε πρόσβαση σε ανεξάρτητα στοιχεία τα οποία δείχνουν την τιμή ενός στοιχείου [1,0] του πρώτου πίνακα:

```
System.Console.Write("{0}", jaggedArray4[0][1, 0]);
```

Η μέθοδος Length επιστρέφει των αριθμό των πινάκων που περιέχονται σε ένα πίνακα jagged. Για παράδειγμα, υποθέτουμε πως έχουμε δηλώσει τον προηγούμενο πίνακα:

```
System.Console.WriteLine(jaggedArray4.Length);
```

Αυτή η γραμμή μας επιστρέφει την τιμή 3.

Το επόμενο παράδειγμα δημιουργεί ένα πίνακα που τα στοιχεία του είναι πίνακες και κάθε στοιχείο του έχει διαφορετικό μέγεθος:

```
class ArrayTest
{
   static void Main()
   {
      // Declare the array of two elements:
      int[][] arr = new int[2][];
```

```
        // Initialize the elements:
        arr[0] = new int[5] { 1, 3, 5, 7, 9 };
        arr[1] = new int[4] { 2, 4, 6, 8 };

        // Display the array elements:
        for (int i = 0; i < arr.Length; i++)
        {
            System.Console.Write("Element({0}): ", i);

            for (int j = 0; j < arr[i].Length; j++)
            {
                System.Console.Write("{0}{1}", arr[i][j], j == (arr[i].Length - 1) ? "" : " ");
            }
            System.Console.WriteLine();
        }
        // Keep the console window open in debug mode.
        System.Console.WriteLine("Press any key to exit.");
        System.Console.ReadKey();
    }
}
/* Output:
    Element(0): 1 3 5 7 9
    Element(1): 2 4 6 8
*/
```

Τα παρακάτω παραδείγματα δημιουργούν πίνακες: ενιαίας διάστασης, πολυδιάστατους και jagged:

```
class TestArraysClass
{
    static void Main()
    {
        // Declare a single-dimensional array
        int[] array1 = new int[5];

        // Declare and set array element values
        int[] array2 = new int[] { 1, 3, 5, 7, 9 };

        // Alternative syntax
        int[] array3 = { 1, 2, 3, 4, 5, 6 };

        // Declare a two dimensional array
        int[,] multiDimensionalArray1 = new int[2, 3];

        // Declare and set array element values
```

```
        int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };

        // Declare a jagged array
        int[][] jaggedArray = new int[6][];

        // Set the values of the first array in the jagged array structure
        jaggedArray[0] = new int[4] { 1, 2, 3, 4 };
    }
}
```

3.3 Κληρονομικότητα

3.3.1 Κλάσεις και αντικείμενα

Στον αντικειμενοστραφή προγραμματισμό, η κλάση είναι μια κατασκευή που χρησιμοποιείται ως πρότυπο σχεδίασης για τη δημιουργία αντικειμένων της κλάσης. Αυτό το πρότυπο σχεδιασμού περιγράφει τις καταστάσεις και συμπεριφορές που μοιράζονται όλα τα αντικείμενα μιας κλάσης. Ένα αντικείμενο μιας κλάσης που έχει δωθεί ονομάζεται instance της κλάσης. Η κλάση που περιέχει αυτό το instance μπορεί να θεωρείται ως ο τύπος αυτού του αντικειμένου, π.χ. ένα αντικείμενο instance της κλάσης «Φρούτο» θα ήταν τύπου «φρούτο».

Μια κλάση συνήθως αντιπροσωπεύει ένα ουσιαστικό, όπως ένα πρόσωπο, μέρος ή πράγμα – είναι ένα πρότυπο μιας έννοιας στο πλαίσιο ενός προγράμματος ηλεκτρονικού υπολογιστή. Βασικά, ενσωματώνει την κατάσταση και την συμπεριφορά της έννοιας που αντιπροσωπεύει. Ενσωματώνει την έννοια της κατάστασης μέσα από placeholders δεδομένων που λέγονται γνωρίσματα(ή member variables ή instance variables). Ενσωματώνει την έννοια της συμπεριφοράς μέσα από επαναχρησιμοποιούμενα τμήματα κώδικα που ονομάζονται μέθοδοι.

Η κλάση είναι ένα συνεκτικό πακέτο που αποτελείται από ένα συγκεκριμένο είδος μεταδεδομένων. Η κλάση έχει και διεπαφή και δόμηση. Η διεπαφή περιγράφει πως αλληλεπιδρούν οι μέθοδοι με τις κλάσεις και με τα instances αυτής, ενώ η δόμηση περιγράφει πως τα δεδομένα τμηματοποιούνται σε γνωρίσματα μέσα μια instance. Μια κλάση μπορεί επίσης να έχει μια αναπαράσταση στο χρόνο εκτέλεσης, που παρέχει υποστήριξη χρόνου εκτέλεσης για το χειρισμό των συσχετισμένων, με τις κλάσεις, μεταδεδομένων.

Παράδειγμα κλάσεων:

```
using System;

public class Program
{
   public static void Main(string[] args)
   {
      Console.WriteLine("Hello world!");
   }
}
```

Υπάρχουν δύο είδη κλάσεων: οι ενσωματωμένες κλάσεις που εμφανίζονται με το .NET Framework και ονομάζονται Framework Class Library και οι καθορισμένες από το χρήστη κλάσεις, τις οποίες δημιουργούν οι ίδιοι οι χρήστες.

Οι κλάσεις περιέχουν δεδομένα (με την μορφή ιδιοτήτων και μεταβλητών) και συμπεριφορές (με την μορφή μεθόδων για την επεξεργασία των δεδομένων). Όταν ορίζουμε μια μεταβλητή σε μια κλάση την ονομάζουμε member variable ή instance variable. Η ονομασία instance προέρχεται από το γεγονός ότι όταν πάμε να δημιουργήσουμε ένα αντικείμενο, υποδεικνύουμε σε μια κλάση να δημιουργήσει το αντικείμενο αυτό, έτσι το instance μιας κλάσης είναι το αντικείμενο μιας κλάσης και instance variable είναι η μεταβλητή που υπάρχει μέσα στην κλάση.

Οι κλάσεις δηλώνονται με την λέξη κλειδί class, ως εξής;

```
[attributes] [modifiers] class identifier [:base-list] { class-body }[;]
```

Όπου:

Attributes (προαιρετικά): επιπρόσθετη πληροφορία δήλωσης

Modifiers (προαιρετικά): τα επιτρεπόμενα modifiers είναι new, abstract, sealed και τα τέσσερα modifiers πρόσβασης.

Identifier: το όνομα της κλάσης

Base-list (προαιρετική): μια λίστα που περιέχει την base class και οποιαδήποτε εφαρμοσμένη διεπαφή, όπου όλες διαχωρίζονται με κόμμα.

Class-body: δήλωση των μελών της κλάσης

Ένα αντικείμενο είναι μια οντότητα που μπορεί να χειριστεί από τις εντολές μιας γλώσσας προγραμματισμού, όπως η τιμή, η μεταβλητή, η συνάρτηση ή η δομή δεδομένων.

Όσον αφορά τον αντικειμενοστραφή προγραμματισμό, ως αντικείμενο θεωρείται μια συλλογή γνωρισμάτων (στοιχεία αντικειμένου) και συμπεριφορών (μέθοδοι ή υπορουτίνες) που ενσωματώνονται σε μια οντότητα. Έτσι, ενώ απλοί τύποι δεδομένων είναι απλώς κομμάτια πληροφοριών, τα αντικειμενοστραφή αντικείμενα είναι πολύπλοκοι τύποι που έχουν πολλαπλά κομμάτια πληροφοριών και συγκεκριμένες ιδιότητες. Αντί απλά να αντιστοιχηθεί μια τιμή, όπως int=10, τα αντικείμενα πρέπει να «κατασκευαστούν». Στον πραγματικό κόσμο, εάν ένα όπλο(π.χ. ένα Colt45) είναι ένα «αντικείμενο», οι φυσικές του ιδιότητες και η λειτουργία του να πυροβολεί έχουν προσδιοριστεί μεμονωμένα. Αφού οι ιδιότητες του «αντικειμένου» Colt45 έχουν προσδιοριστή στην μορφή μιας κλάσης(ας την ονομάσουμε «όπλο»), μπορεί να αντιγράφεται ασταμάτητα για να δημιουργηθούν πανομοιότυπα αντικείμενα που μοιάζουν και λειτουργούν με τον ίδιο ακριβώς τρόπο.

Τα αντικείμενα είναι το θεμέλιο του αντικειμενοστραφούς προγραμματισμού, και υπάρχουν θεμελιώδεις τύποι δεδομένων στις αντικειμενοστραφείς γλώσσες προγραμματισμού. Αυτές οι γλώσσες προγραμματισμού παρέχουν εκτενή σημασιολογική και συντακτική υποστήριξη για το χειρισμό των αντικειμένων, συμπεριλαμβανομένου ενός ιεραρχικού συστήματος τύπων, ειδική σημείωση για τη δήλωση και την κλήση των μεθόδων, καθώς και εγκαταστάσεις για την απόκρυψη επιλεγμένων πεδίων από τους προγραμματιστές client. Εντούτοις, τα αντικείμενα και ο αντικειμενοστραφής προγραμματισμός μπορεί να εφαρμοστεί σε οποιαδήποτε γλώσσα.

Τα αντικείμενα είναι προγραμματιστικές κατασκευές τα οποία έχουν δεδομένα, συμπεριφορές και ταυτότητα. Τα δεδομένα των αντικειμένων εμπεριέχονται στα πεδία, στις ιδιότητες και γεγονότων των αντικειμένων. Οι συμπεριφορές των αντικειμένων καθορίζονται από τις διεπαφές τους.

Το παρακάτω παράδειγμα δείχνει πως οι μεταβλητές ενός τύπου αντικειμένου αποδέχονται τιμές οποιουδήποτε τύπου δεδομένων και πως οι μεταβλητές του τύπου αντικειμένου χρησιμοποιεί μεθόδους σε αντικείμενα από το .NET Framework.

```
class ObjectTest
{
  public int i = 10;
}

class MainClass2
{
  static void Main()
  {
    object a;
    a = 1;   // an example of boxing
    Console.WriteLine(a);
    Console.WriteLine(a.GetType());
    Console.WriteLine(a.ToString());

    a = new ObjectTest();
```

```
    ObjectTest classRef;
    classRef = (ObjectTest)a;
    Console.WriteLine(classRef.i);
  }
}
/* Output
   1
   System.Int32
   1
 * 10
*/
```

Τα αντικείμενα έχουν ταυτότητα. Δύο αντικείμενα με το ίδιο σύνολο δεδομένων δεν σημαίνει πως είναι το ίδιο αντικείμενο.

Τα αντικείμενα έχουν τις παρακάτω ιδιότητες:

- Ο,τι χρησιμοποιούν οι χρήστες στη C# είναι αντικείμενο, συμπεριλαμβανομένων των Windows Forms και των ελέγχων.
- Τα αντικείμενα έχουν υπόσταση: δημιουργούνται από πρότυπα και ορίζονται από κλάσεις και δομές.
- Χρησιμοποιούν τις ιδιότητες για να αποκτήσουν και να αλλάξουν τα δεδομένα που περιέχουν.
- Το Visual Studio παρέχει εργαλεία για τη διαχείριση των αντικειμένων: το properties window επιτρέπει στους χρήστες να αλλάξουν χαρακτηριστικά των αντικειμένων όπως το Widows Forms. Ο Object Browser επιτρέπει στους χρήστες να ελέγξουν τα περιεχόμενα ενός αντικειμένου.

3.3.2 Structs και Constructors

Μια δομή, είναι ένας δομημένος τύπος που συναθροίζει ένα σταθερό σύνολο αντικειμένων με επισημάνσεις, πιθανότατα διαφορετικών τύπων, σε ένα ενιαίο αντικείμενο.

Οι δομές (structs) ορίζονται με την χρήση της λέξης κλειδί struct, για παράδειγμα:

```
public struct PostalAddress
{
   // Fields, properties, methods and events go here...
}
```

Ένας τύπος δομής είναι ένας value type που χρησιμοποιείται συνήθως για να συμπεριλάβει μικρές ομάδες συσχετιζόμενων μεταβλητών, όπως οι συντεταγμένες ενός ορθογωνίου ή τα χαρακτηριστικά ενός στοιχείου σε ένα κατάλογο. Το παρακάτω παράδειγμα δείχνει μια απλή δήλωση δομής:

```
public struct Book
{
   public decimal price;
   public string title;
   public string author;
}
```

Το παρακάτω παράδειγμα δηλώνει μια δομή με τρία μέλη:μια ιδιότητα, μια μέθοδο και ένα ιδιωτικό πεδίο. Δημιουργεί ένα instance της δομής και το βάζει σε χρήση:

```
// struct1.cs
using System;
struct SimpleStruct
{
   private int xval;
   public int X
   {
      get
```

```
        {
            return xval;
        }
        set
        {
            if (value < 100)
                xval = value;
        }
    }
    public void DisplayX()
    {
        Console.WriteLine("The stored value is: {0}", xval);
    }
}

class TestClass
{
    public static void Main()
    {
        SimpleStruct ss = new SimpleStruct();
        ss.X = 5;
        ss.DisplayX();
    }
}
```

το αποτέλεσμα που παίρνουμε είναι:

```
The stored value is: 5
```

Οι δομές μπορούν επίσης να περιέχουν constructors, σταθερές, πεδία, μεθόδους, τελεστές, γεγονότα, ευρετήρια(indexers) και εμφωλευμένους τύπους. Εάν απαιτείται μια σειρά από τόσα πολλά μέλη, οι χρήστες μπορούν να τροποποιήσουν τον τύπο σε κλάση αντί για δομή.

Οι δομές μοιράζονται κατά κύριο λόγο την ίδια σύνταξη με τις κλάσεις, παρόλο που οι δομές είναι πιο περιορισμένες από τις κλάσεις.

- Χωρίς τη δήλωση δομών, τα πεδία δεν μπορούν να αρχικοποιηθούν αν δεν οριστούν ως στατικά ή σταθερά.
- Μια δομή μπορεί να μη δηλώσει ένα προκαθορισμένο constructor (constructor χωρίς παραμέτρους) ή destructor
- Οι δομές αντιγράφονται κατά την αντιστοίχηση. Όταν μια δομή αντιστοιχίζεται σε μια μεταβλητή, όλα τα δεδομένα αντιγράφονται, και οποιαδήποτε τροποποίηση στο νέο αντίγραφο δεν αλλάζει τα δεδομένα του αυθεντικού αντίγραφου.
- Οι δομές είναι value types και οι κλάσεις reference types.
- Αντίθετα από τις κλάσεις, οι δομές μπορούν να γίνουν instantiated χωρίς την χρήση του τελεστή new.
- Οι δομές μπορούν να δηλώσουν constructors που έχουν παραμέτρους
- Μια δομή δεν μπορεί να κληρονομήσει από άλλη δομή ή κλάση αλλά δεν μπορεί να γίνει η βάση μιας κλάσης. Όλες οι δομές κληρονομούν απευθείας από το System.ValueType το οποίο κληρονομεί από το System.Object
- Μια δομή μπορεί να εφαρμόσει διεπαφές, αλλά δεν μπορεί να κληρονομήσει από άλλη δομή. Για αυτό τον λόγο τα μέλη των δομών δεν μπορούν να δηλωθούν ως προστατευμένα.
- Μια δομή μπορεί να χρησιμοποιηθεί ως μηδενικός τύπος και μπορεί να αντιστοιχηθεί με μηδενική τιμή.

Στον αντικειμενοστραφή προγραμματισμό, ένας constructor σε μια κλάση είναι ένας ειδικός τύπος μιας υπό-ρουτίνας που καλείται κατά τη δημιουργία ενός αντικειμένου. Προετοιμάζει το νέο αντικείμενο για χρήση, αποδέχοντας συχνά παραμέτρους τις οποίες χρησιμοποιεί ο constructor για να τοποθετήσει τις όποιες απαραίτητες μεταβλητές μελών, όταν δημιουργείται το αντικείμενο.

Ένας constructor μοιάζει με μια μέθοδο, αλλά διαφέρει από αυτή στο ότι δεν έχει ποτέ ρητή εντολή επιστροφής τύπου, δεν κληρονομείται και συνήθως έχει διαφορετικούς κανόνες για τους τροποποιητές πεδίων. Οι constructors συνήθως

συγχέονται επειδή έχουν το ίδιο όνομα με την κλάση που δηλώνεται. Το καθήκον τους είναι να αρχικοποιούν τα δεδομένα μελών των αντικειμένων και να εγκαθιδρύουν τα αμετάβλητα της κλάσης, κάτι που αποτυγχάνει αν τα αμετάβλητα δεν είναι έγκυρα. Ένας σωστά γραμμένος constructor αφήνει το αντικείμενο στην κατάσταση «έγκυρο».

Παράδειγμα Constructor:

```
public class MyClass
{
  private int a;
  private string b;

  //constructor
  public MyClass(int a, string b)
  {
    this.a = a;
    this.b = b;
  }
}
//code somewhere
//instantiating an object with the constructor above
MyClass c = new MyClass(42, "string");
```

Static constructor

Ένας στατικός constructor αρχικοποιεί στατικά δεδομένα. Οι στατικοί constractors επιτρέπουν την αρχικοποίηση σύνθετων στατικών μεταβλητών. Οι στατικοί constractors μπορούν να κληθούν μια φορά και η κλήση μπορεί να γίνει σιωπηλά από το χρόνο εκτέλεσης ακριβώς πριν γίνει προσβάσιμη η κλάση για πρώτη φορά. Οποιαδήποτε κλήση στην κλάση ενεργοποιεί την εκτέλεση του στατικού constructor.

Παράδειγμα στατικού constructor:

```
public class MyClass
{
  private static int _A;

  //normal constructor
  static MyClass()
  {
    _A = 32;
  }

  //standard default constructor
  public MyClass()
  {

  }
}
//code somewhere
//instantiating an object with the constructor above
//right before the instantiation
//the variable static constructor is executed and _A is 32
MyClass c = new MyClass();
```

3.3.3 Έννοια Κληρονομικότητας

Η κληρονομικότητα είναι ένας τρόπος για να δημιουργούνται νέες κλάσεις με την χρήση κλάσεων που έχουν ήδη οριστεί. Η κληρονομικότητα χρησιμοποιείται για να βοηθήσει στην επαναχρησιμοποίηση του υφιστάμενου κώδικα με λίγη ή χωρίς καθόλου τροποποίηση. Οι νέες κλάσεις γνωστές και ως υπό – κλάσεις (ή εξαγόμενες κλάσεις), κληρονομούν γνωρίσματα και συμπεριφορές από τις κλάσεις που προϋπάρχουν, γνωστές και ως υπέρ – κλάσεις. Η κληρονομική σχέση μεταξύ των κλάσεων αυτών δημιουργεί μια ιεραρχία.

Η κληρονομικότητα δεν πρέπει να συγχέεται με τον πολυμορφισμό (έννοια που εξετάζεται παρακάτω). Η κληρονομικότητα είναι μια σχέση μεταξύ εφαρμογών ενώ ο πολυμορφισμός υπό-τύπων(Subtype) είναι μια σχέση μεταξύ τύπων. Σε κάποιες

γλώσσες προγραμματισμού οι έννοιες συμπίπτουν, διότι ο μόνος τρόπος για να δηλωθεί ένας υπό- τύπος είναι να οριστεί μια νέα κλάση που να κληρονομεί την εφαρμογή σε μια άλλη κλάση. Είναι δυνατόν να εξαχθεί μια κλάση της οποίας το αντικείμενο συμπεριφέρεται με λανθάνοντα τρόπο όταν χρησιμοποιείται σε ένα πλαίσιο όπου αναμένεται η κλάση γόνος. Για αυτό και λέμε πως η κληρονομικότητα δεν συνεπάγεται ότι οι υπό- τύποι θα συμπεριφέρονται με σωστό τρόπο.

Μια κλάση μπορεί να κληρονομήσει από μια άλλη κλάση. Αυτό επιτυγχάνεται με την προσθήκη μιας άνω κάτω τελείας μετά το όνομα της κλάσης όταν δηλώνεται η κλάση, και βάζοντας το όνομα της κλάσης που θα κληρονομήσει μετά την άνω κάτω τελεία:

```
public class A
{
   public A() { }
}

public class B : A
{
   public B() { }
}
```

Η νέα κλάση παίρνει όλα τα μη ιδιωτικά δεδομένα και συμπεριφορές της πρότυπης κλάσης, συν τα όποια άλλα δεδομένα και συμπεριφορές, που έχει ορίσει η ίδια για τον εαυτό της. Η κλάση αυτή έχει δύο νέους τύπους: τον τύπο της νέας κλάσης και τον τύπο της κλάσης που κληρονομεί.

Στο παραπάνω παράδειγμα η κλάση B μπορεί να είναι αποδοτικά και η κλάση A και η B. Όταν αποκτούμε πρόσβαση σε ένα αντικείμενο κλάσης B, μπορούμε να χρησιμοποιήσουμε την λειτουργία cast για να το μετατρέψουμε σε αντικείμενο A. Το αντικείμενο B δεν τροποποιείται από την cast, αλλά η δυνατότητα να δούμε το αντικείμενο B περιορίζεται σε δεδομένα και συμπεριφορές του αντικειμένου A. μετά την τροποποίηση του B σε A, το A μπορεί να τροποποιηθεί πάλι σε B.Δεν μπορούν όλες οι instances του A να τροποποιηθούν σε αυτές του B – μόνο αυτές που είναι πράγματι instances του B. εάν έχουμε πρόσβαση σε μια κλάση B ως τύπος B, λαμβάνουμε τους τύπους και τις συμπεριφορές και της κλάσης A και της B.

3.3.4 Πολυμορφισμός

Ο πολυμορφισμός υπό – τύπων, που είναι γνωστός απλά και ως πολυμορφισμός στα πλαίσια του αντικειμενοστραφούς προγραμματισμού, είναι η ικανότητα ενός τύπου A, να παρουσιάζεται και να χρησιμοποιείται ως ένας άλλος τύπος B. Ο σκοπός του πολυμορφισμού είναι να εφαρμόζει ένα τρόπο προγραμματισμού που λέγεται message-passing, στον οποίο αντικείμενα διαφόρων τύπων ορίζουν μια κοινή διεπαφή λειτουργιών για τους χρήστες.

Η κύρια χρήση του πολυμορφισμού σύμφωνα με τη θεωρία του αντικειμενοστραφούς προγραμματισμού, είναι η ικανότητα των αντικειμένων να ανήκουν σε διαφορετικούς τύπους, να ανταποκρίνονται σε μια μέθοδο, σε ένα πεδίο ή σε ιδιότητες κλήσεις της ίδιας ονομασίας και κάθε ένας από τους τύπους να ανταποκρίνεται σε μια συμπεριφορά συγκεκριμένου τύπου. Ο προγραμματιστής δεν χρειάζεται να γνωρίζει τον ακριβή τύπο του αντικειμένου εκ των προτέρων και έτσι η ακριβής συμπεριφορά καθορίζεται κατά τον χρόνο εκτέλεσης.

Η ιδανική εφαρμογή πολυμορφισμού στη γλώσσα C# είναι η χρήση των αποθηκευμένων διεπαφών σε μια κοινή γλώσσα API η οποία δεν έχει καμία εξάρτηση από τον κώδικα χρήστη:

```
// Assembly: Common Classes
namespace CommonClasses
{
   public interface IAnimal
   {
      string Name { get; }
      string Talk();
   }
}

// Assembly: Animals
using System;
using CommonClasses;

namespace Animals
```

```
{
   public abstract class AnimalBase
   {
      public string Name { get; private set; }

      protected AnimalBase(string name) {
         Name = name;
      }
   }

   public class Cat : AnimalBase, IAnimal
   {
      public Cat(string name) : base(name) {
      }

      public string Talk() {
         return "Meowww!";
      }
   }

   public class Dog : AnimalBase, IAnimal
   {
      public Dog(string name) : base(name) {
      }

      public string Talk() {
         return "Arf! Arf!";
      }
   }
}


// Assembly: Program
// References and Uses Assemblies: Common Classes, Animals
using System;
using System.Collections.Generic;
using Animals;
using CommonClasses;

namespace Program
{
   public class TestAnimals
   {
      // prints the following:
      //   Missy: Meowww!
      //   Mr. Mistoffelees: Meowww!
      //   Lassie: Arf! Arf!
      //
      public static void Main(string[] args)
```

```
    {
        var animals = new List<IAnimal>() {
            new Cat("Missy"),
            new Cat("Mr. Mistoffelees"),
            new Dog("Lassie")
        };

        foreach (var animal in animals) {
            Console.WriteLine(animal.Name + ": " + animal.Talk());
        }
    }
  }
}
```

3.4 Προχωρημένα χαρακτηριστικά

3.4.1 Διεπαφές

Οι διεπαφές περιγράφουν μια ομάδα συσχετισμένων λειτουργικοτήτων που ανήκουν σε οποιαδήποτε κλάση ή δομή. Οι διεπαφές μπορεί να συνίστανται από μεθόδους, ιδιότητες, γεγονότα, ευρετήρια ή από οποιονδήποτε συνδυασμό αυτών των τεσσάρων τύπων μελών. Μια διεπαφή δεν μπορεί να εμπεριέχει πεδία. Τα μέλη των διεπαφών κοινοποιούνται αυτόματα.

Οι διεπαφές ορίζονται με την χρήση της λέξεως κλειδί interface, όπως στο παρακάτω παράδειγμα:

```
interface IEquatable<T>
{
   bool Equals(T obj);
}
```

Όταν λέμε ότι μια κλάση ή δομή κληρονομεί μια διεπαφή, εννοούμε ότι η κλάση ή η δομή παρέχει μια εφαρμογή για όλα τα μέλη που έχουν οριστεί από την

διεπαφή. Η διεπαφή από μόνη της δεν παρέχει καμία λειτουργικότητα ότι θα μπορεί η κλάση να κληρονομήσει με τον ίδιο τρόπο που κληρονομείται μια λειτουργικότητα της πρότυπης κλάσης. Εντούτοις, εάν η πρότυπη κλάση εφαρμόσει μια διεπαφή, η εξαγόμενη κλάση κληρονομεί αυτή την εφαρμογή.

Οι κλάσεις και οι δομές μπορούν να κληρονομήσουν από τις διεπαφές με ένα παρόμοιο τρόπο με αυτόν που κληρονομούν οι κλάσεις μια πρότυπη κλάση ή δομή, με δύο εξαιρέσεις:

- Η κλάση ή η δομή μπορεί να κληρονομήσει περισσότερες από μια διεπαφές.
- Όταν μια κλάση ή μια δομή κληρονομεί μια διεπαφή, κληρονομεί μόνο τα ονόματα των μεθόδων και τις υπογραφές, επειδή η διεπαφή από μόνη της δεν περιέχει εφαρμογές. Για παράδειγμα:

```
public class Car : IEquatable<Car>
{
   public string Make {get; set;}
   public string Model { get; set; }
   public string Year { get; set; }

   // Implementation of IEquatable<T> interface
   public bool Equals(Car car)
   {
      if (this.Make == car.Make &&
         this.Model == car.Model &&
         this.Year == car.Year)
      {
         return true;
      }
      else
         return false;
   }
}
```

Για να εφαρμοστεί ένα μέλος μιας διεπαφής, το αντίστοιχο μέλος στην κλάση πρέπει να είναι public(κοινόχρηστο), μη – στατικό και να έχει το ίδιο όνομα και την

ίδια υπογραφή με το μέλος της διεπαφής. Οι ιδιότητες και τα ευρετήρια σε μια κλάση μπορούν να ορίσουν επιπλέον accessors για μια ορισμένη ιδιότητα ή για ένα ορισμένο ευρετήριο σε μια διεπαφή. Για παράδειγμα, μια διεπαφή μπορεί να δηλώνει μια ιδιότητα με ένα get accessor, αλλά η κλάση στην οποία εφαρμόζεται η διεπαφή να δηλώνει την ίδια ιδιότητα με ένα accessor get αλλά και με ένα accessor set. Παρόλα αυτά, εάν η ιδιότητα ή το ευρετήριο χρησιμοποιεί ρητή εφαρμογή, οι accessor θα πρέπει να ταιριάζουν.

Οι διεπαφές και τα μέλη των διεπαφών είναι αφαιρούμενα. Οι διεπαφές δεν παρέχουν προκαθορισμένη εφαρμογή.

Η διεπαφή IEquatable<T> ανακοινώνει στον χρήστη του αντικειμένου ότι το αντικείμενο μπορεί να ορίσει αν είναι ισότιμο με άλλα αντικείμενα του ίδιου τύπου και έτσι ο χρήστης της διεπαφής δε χρειάζεται να γνωρίζει πως αυτό θα εφαρμοστεί.

Οι διεπαφές μπορούν να κληρονομήσουν άλλες διεπαφές. Είναι δυνατό για μια κλάση να κληρονομήσει μια διεπαφή πολλές φορές, μέσω πρότυπων κλάσεων ή διεπαφών που αυτή κληρονομεί. Σε αυτή την περίπτωση, η κλάση μπορεί να εφαρμόζει μια διεπαφή την φορά, εάν αυτή έχει δηλωθεί ως τμήμα της νέας κλάσης. Εάν η κληρονομούμενη διεπαφή δεν έχει δηλωθεί ως τμήμα της νέας κλάσης, η εφαρμογή της παρέχεται από την πρότυπη κλάση από την οποία δηλώθηκε. Είναι δυνατό για μια πρότυπη κλάση να εφαρμόσει μέλη διεπαφών χρησιμοποιώντας εικονικά μέλη.

Μια διεπαφή έχει τις παρακάτω ιδιότητες:

- Οποιοσδήποτε μη αφαιρούμενος τύπος που κληρονομεί τη διεπαφή πρέπει να εφαρμόζει όλα τα μέλη του.

- Οι διεπαφές μπορούν να περιέχουν γεγονότα, ευρετήρια, μεθόδους και ιδιότητες
- Οι διεπαφές δε διαθέτουν εφαρμογές για τις μεθόδους.
- Οι κλάσεις και οι δομές μπορούν να κληρονομήσουν περισσότερες από μια διεπαφές
- Μια διεπαφή μπορεί από μόνη της να κληρονομήσει, από πολλαπλές διεπαφές.

Στο παρακάτω παράδειγμα η κλάση ImplementationClass πρέπει να εφαρμόσει μια μέθοδο που λέγεται SampleMethod, η οποία δεν έχει παραμέτρους και δεν επιστρέφει τιμή:

```
interface ISampleInterface
{
   void SampleMethod();
}

class ImplementationClass : ISampleInterface
{
   // Explicit interface member implementation:
   void ISampleInterface.SampleMethod()
   {
      // Method implementation.
   }

   static void Main()
   {
      // Declare an interface instance.
      ISampleInterface obj = new ImplementationClass();

      // Call the member.
      obj.SampleMethod();
   }
}
```

Στο επόμενο παράδειγμα έχουμε μια εφαρμογή διεπαφής. Η διεπαφή περιέχει τη δήλωση της ιδιότητας και η κλάση την εφαρμογή. Οποιαδήποτε instance μιας κλάσης που εφαρμόζει το IPoint έχει ακέραιες ιδίοτητες x,y.:

```
interface IPoint
   {
     // Property signatures:
     int x
     {
       get;
       set;
     }

     int y
     {
       get;
       set;
     }
   }

   class Point : IPoint
   {
     // Fields:
     private int _x;
     private int _y;

     // Constructor:
     public Point(int x, int y)
     {
       _x = x;
       _y = y;
     }

     // Property implementation:
     public int x
     {
       get
       {
         return _x;
       }

       set
       {
         _x = value;
       }
     }

     public int y
```

```
    {
      get
      {
        return _y;
      }
      set
      {
        _y = value;
      }
    }
  }

  class MainClass
  {
    static void PrintPoint(IPoint p)
    {
      Console.WriteLine("x={0}, y={1}", p.x, p.y);
    }

    static void Main()
    {
      Point p = new Point(2, 3);
      Console.Write("My Point: ");
      PrintPoint(p);
    }
  }
  // Output: My Point: x=2, y=3
```

3.4.2 Overloading

3.4.2.1 Function or Method overloading

Function ή method overloading είναι ένα χαρακτηριστικό που υπάρχει σε διάφορες προγραμματιστικές γλώσσες όπως η C#, το οποίο επιτρέπει τη δημιουργία μιας σειράς από μεθόδους με το ίδιο όνομα, οι οποίες διαφέρουν η μία από την άλλη στον τύπο εισαγωγής και εξαγωγής της συνάρτησης.

Για παράδειγμα, η doTask() και η doTask(object O) είναι overloaded methods. Για να κληθεί η τελευταία, ένα αντικείμενο πρέπει να περαστεί ως παράμετρος, ενώ η

πρώτη δεν απαιτεί παράμετρο και καλείται με κενό το πεδίο της παραμέτρου. Ένα κοινό σφάλμα θα ήταν, στη δεύτερη μέθοδο, να αντιστοιχηθεί μια προκαθορισμένη τιμή στο αντικείμενο, κάτι που θα προκαλούσε ένα λάθος ασαφούς κλήσης(ambiguous call), καθώς ο μεταγλωττιστής δε θα ήξερε ποια μέθοδο να χρησιμοποιήσει.

Η method overloading συνήθως συσχετίζεται με γλώσσες προγραμματισμού στατικού τύπου που ενισχύει τον έλεγχο των τύπων στις κλήσεις συναρτήσεων. Όταν μια μέθοδος γίνεται overloading, δημιουργείται ένας αριθμός διαφορετικών μεθόδων που τυγχάνει να έχουν το ίδιο όνομα. Το πρόβλημα του ποια από τις μεθόδους θα χρησιμοποιηθεί, επιλύεται στο χρόνο μεταγλώττισης.

3.4.2.2 Operator Overloading

Το operator overloading επιτρέπει τις εφαρμογές τελεστών που είναι καθορισμένες από τους χρήστες να εξειδικεύονται για λειτουργίες, όπου ο ένας ή και οι δύο τελεσταίοι είναι τύπου κλάσης ή δομής, η οποία καθορίζεται από τον χρήστη.

Το παρακάτω παράδειγμα δείχνει πως χρησιμοποιείται το operator overloading για να δημιουργήσει μια σύνθετη αριθμητική κλάση Complex η οποία ορίζει την σύνθετη προσθήκη(complex addition):

```
public struct Complex
{
   public int real;
   public int imaginary;

   public Complex(int real, int imaginary)  //constructor
   {
      this.real = real;
      this.imaginary = imaginary;
   }

   // Declare which operator to overload (+),
   // the types that can be added (two Complex objects),
```

```
    // and the return type (Complex):
    public static Complex operator +(Complex c1, Complex c2)
    {
        return new Complex(c1.real + c2.real, c1.imaginary + c2.imaginary);
    }

    // Override the ToString() method to display pointA complex number in the
traditional format:
    public override string ToString()
    {
        return (System.String.Format("{0} + {1}radius", real, imaginary));
    }
}

class TestComplex
{
    static void Main()
    {
        Complex num1 = new Complex(2, 3);
        Complex num2 = new Complex(3, 4);

        // Add two Complex objects through the overloaded plus operator:
        Complex sum = num1 + num2;

        // Print the numbers and the sum using the overriden ToString method:
        System.Console.WriteLine("First complex number:  {0}", num1);
        System.Console.WriteLine("Second complex number: {0}", num2);
        System.Console.WriteLine("The sum of the two numbers: {0}", sum);

        // Keep the console window open in debug mode.
        System.Console.WriteLine("Press any key to exit.");
        System.Console.ReadKey();
    }
}
/* Output:
    First complex number:  2 + 3i
    Second complex number: 3 + 4i
    The sum of the two numbers: 5 + 7i
*/
```

Αυτό το παράδειγμα χρησιμοποιείται για να δείξει πως το operator overloading μπορεί να εφαρμόσει ένα three-valued logical τύπο. Οι πιθανές τιμές του τύπου είναι DBBool.dbTrue, DBBool.dbFalse, and DBBool.dbNull, όπου το dbNull μέλος δείχνει μια άγνωστη τιμή:

```
// dbbool.cs
using System;

public struct DBBool
{
   // The three possible DBBool values:
   public static readonly DBBool dbNull = new DBBool(0);
   public static readonly DBBool dbFalse = new DBBool(-1);
   public static readonly DBBool dbTrue = new DBBool(1);
   // Private field that stores -1, 0, 1 for dbFalse, dbNull, dbTrue:
   int value;

   // Private constructor. The value parameter must be -1, 0, or 1:
   DBBool(int value)
   {
      this.value = value;
   }

   // Implicit conversion from bool to DBBool. Maps true to
   // DBBool.dbTrue and false to DBBool.dbFalse:
   public static implicit operator DBBool(bool x)
   {
      return x? dbTrue: dbFalse;
   }

   // Explicit conversion from DBBool to bool. Throws an
   // exception if the given DBBool is dbNull, otherwise returns
   // true or false:
   public static explicit operator bool(DBBool x)
   {
      if (x.value == 0) throw new InvalidOperationException();
      return x.value > 0;
   }

   // Equality operator. Returns dbNull if either operand is dbNull,
   // otherwise returns dbTrue or dbFalse:
   public static DBBool operator ==(DBBool x, DBBool y)
   {
      if (x.value == 0 || y.value == 0) return dbNull;
      return x.value == y.value? dbTrue: dbFalse;
   }

   // Inequality operator. Returns dbNull if either operand is
   // dbNull, otherwise returns dbTrue or dbFalse:
   public static DBBool operator !=(DBBool x, DBBool y)
   {
      if (x.value == 0 || y.value == 0) return dbNull;
```

```
    return x.value != y.value? dbTrue: dbFalse;
}

// Logical negation operator. Returns dbTrue if the operand is
// dbFalse, dbNull if the operand is dbNull, or dbFalse if the
// operand is dbTrue:
public static DBBool operator !(DBBool x)
{
    return new DBBool(-x.value);
}

// Logical AND operator. Returns dbFalse if either operand is
// dbFalse, dbNull if either operand is dbNull, otherwise dbTrue:
public static DBBool operator &(DBBool x, DBBool y)
{
    return new DBBool(x.value < y.value? x.value: y.value);
}

// Logical OR operator. Returns dbTrue if either operand is
// dbTrue, dbNull if either operand is dbNull, otherwise dbFalse:
public static DBBool operator |(DBBool x, DBBool y)
{
    return new DBBool(x.value > y.value? x.value: y.value);
}

// Definitely true operator. Returns true if the operand is
// dbTrue, false otherwise:
public static bool operator true(DBBool x)
{
    return x.value > 0;
}

// Definitely false operator. Returns true if the operand is
// dbFalse, false otherwise:
public static bool operator false(DBBool x)
{
    return x.value < 0;
}

// Overload the conversion from DBBool to string:
public static implicit operator string(DBBool x)
{
    return x.value > 0 ? "dbTrue"
        : x.value < 0 ? "dbFalse"
        : "dbNull";
}

// Override the Object.Equals(object o) method:
public override bool Equals(object o)
```

```
  {
    try
    {
      return (bool) (this == (DBBool) o);
    }
    catch
    {
      return false;
    }
  }

  // Override the Object.GetHashCode() method:
  public override int GetHashCode()
  {
    return value;
  }

  // Override the ToString method to convert DBBool to a string:
  public override string ToString()
  {
    switch (value)
    {
      case -1:
        return "DBBool.False";
      case 0:
        return "DBBool.Null";
      case 1:
        return "DBBool.True";
      default:
        throw new InvalidOperationException();
    }
  }
}
```

```
class Test
{
  static void Main()
  {
    DBBool a, b;
    a = DBBool.dbTrue;
    b = DBBool.dbNull;

    Console.WriteLine( "!{0} = {1}", a, !a);
    Console.WriteLine( "!{0} = {1}", b, !b);
    Console.WriteLine( "{0} & {1} = {2}", a, b, a & b);
    Console.WriteLine( "{0} | {1} = {2}", a, b, a | b);
    // Invoke the true operator to determine the Boolean
```

```
    // value of the DBBool variable:
    if (b)
      Console.WriteLine("b is definitely true");
    else
      Console.WriteLine("b is not definitely true");
  }
}
```

Output

```
!DBBool.True = DBBool.False
!DBBool.Null = DBBool.Null
DBBool.True & DBBool.Null = DBBool.Null
DBBool.True | DBBool.Null = DBBool.True
b is not definitely true
```

3.4.3 Εξαιρέσεις

Στη γλώσσα C#, τα σφάλματα στο χρόνο εκτέλεσης του προγράμματος διαδίδονται(are being propagated) μέσω του προγράμματος με την χρήση ενός μηχανισμού που λέγεται εξαιρέσεις. Ένας κώδικας που εντοπίζει ένα σφάλμα ρίχνει(παράγει) μια εξαίρεση και αφού μπορέσει ο κώδικας να διορθώσει το σφάλμα, αλιεύει την εξαίρεση. Το CLR του .NET framework ή ένας κώδικας ενός προγράμματος έχουν τη δυνατότητα να παράγουν μια εξαίρεση. Όταν παραχθεί η εξαίρεση, αυτή μεταδίδει την στοίβα κλήσης εωσότου βρεθεί μια δήλωση catch για την εξαίρεση. Οι εξαιρέσεις που δεν αλιεύονται, χειρίζονται από ένα χειριστή generic exception που παρέχεται από το σύστημα, το οποίο προβάλλει ένα πλαίσιο διαλόγου.

Τα χαρακτηριστικά χειρισμού εξαίρεσης της γλώσσας C# παρέχουν ένα τρόπο για να αντιμετωπίζεται μια οποιαδήποτε μη αναμενόμενη περίπτωση ή μια περίπτωση εξαίρεσης που εμφανίζεται ενώ εκτελείται το πρόγραμμα. Ο χειρισμός εξαίρεσης χρησιμοποιεί τις λέξεις κλειδιά try, catch και finally για να δοκιμάσει ενέργειες που μπορεί να αποτύχουν, για να χειριστεί αποτυχίες και για να αφαιρέσει πόρους έπειτα.

Σε αυτό το παράδειγμα, μια μέθοδος ελέγχει για μια division by zero και βρίσκει το σφάλμα. Δίχως τον χειρισμό εξαίρεσης, το πρόγραμμα θα τερματιζόταν με ένα ***DivideByZeroException was unhandled*** σφάλμα.

```
int SafeDivision(int x, int y)
{
   try
   {
      return (x / y);
   }
   catch (System.DivideByZeroException dbz)
   {
      System.Console.WriteLine("Division by zero attempted!");
      return 0;
   }
}
```

Οι εξαιρέσεις εκπροσωπούνται από τις κλάσεις που εξάγονται από την κλάση Exception. Αυτή η κλάση αναγνωρίζει τον τύπο της εξαίρεσης και περιέχει ιδιότητες οι οποίες περιέχουν λεπτομέρειες για την εξαίρεση. Η δημιουργία μιας εξαίρεσης περιλαμβάνει τη δημιουργία μιας instance μιας εξαίρεσης, που προέρχεται από μια κλάση, την προαιρετική ρύθμιση των ιδιοτήτων της εξαίρεσης, και έπειτα τη δημιουργία(throw) του αντικειμένου με την χρήση της λέξεως κλειδί throw. Για παράδειγμα:

```
class CustomException : Exception
{
   public CustomException(string message)
   {

   }

}
private static void TestThrow()
{
   CustomException ex =
      new CustomException("Custom exception in TestThrow()");

   throw ex;
}
```

Αφού ρίξει την εξαίρεση (ο κωδικός ή το CLR), ο χρόνος εκτέλεσης ελέγχει την τρέχουσα δήλωση για να διαπιστώσει εάν βρίσκεται μέσα σε ένα try block. Εάν συμβαίνει αυτό, οποιαδήποτε catch blocks που είναι συσχετισμένα με το try block

ελέγχονται για να διαπιστωθεί αν μπορούν να συλλάβουν μια εξαίρεση. Τα catch blocks προσδιορίζουν συνήθως τους τύπους των εξαιρέσεων. Εάν ο τύπος του catch block είναι ο ίδιος με αυτόν της εξαίρεσης, ή με μια κλάση της εξαίρεσης, το catch block μπορεί να χειριστεί την μέθοδο. Για παράδειγμα:

```
static void TestCatch()
{
   try
   {
      TestThrow();
   }
   catch (CustomException ex)
   {
      System.Console.WriteLine(ex.ToString());
   }
}
```

Εάν η δήλωση που ρίχνει την εξαίρεση δεν ανήκει στο try block ή αν το try block που την περικλείει δεν έχει catch block με το οποίο να ταιριάζει, ο χρόνος εκτέλεσης ελέγχει την καλούμενη μέθοδο για μια δήλωση try και για catch blocks. Ο χρόνος εκτέλεσης συνεχίζει την μετάδοση της στοίβας κλήσης, αναζητώντας ένα συμβατό catch block. Αφού βρεθεί το catch block και εκτελεστεί, ο έλεγχος περνά στην επόμενη δήλωση.

Μια δήλωση try μπορεί να εμπεριέχει περισσότερα από ένα catch block. Η πρώτη δήλωση catch που μπορεί να χειριστεί την εξαίρεση, εκτελείται. Οποιαδήποτε δήλωση ακολουθεί, ακόμα και αν είναι συμβατή, αγνοείται. Συνεπώς, τα catch blocks θα πρέπει να διατάσσονται από τα περισσότερο ειδικά στα λιγότερο ειδικά. Για παράδειγμα:

```
static void TestCatch2()
{
   System.IO.StreamWriter sw = null;
   try
   {
      sw = new System.IO.StreamWriter(@"C:\test\test.txt");
      sw.WriteLine("Hello");
   }

   catch (System.IO.FileNotFoundException ex)
   {
      System.Console.WriteLine(ex.ToString());   // put the more specific exception
first
   }
```

```
    catch (System.IO.IOException ex)
    {
        System.Console.WriteLine(ex.ToString());  // put the less specific exceptions last
    }
    finally
    {
        sw.Close();
    }

    System.Console.WriteLine("Done");   // this statement is executed after the catch block
}
```

Πριν εκτελεστεί το catch block, ο χρόνος εκτέλεσης ελέγχει για την ύπαρξη finally blocks. Αυτά επιτρέπουν στον προγραμματιστή να αφαιρέσει την όποια αμφίβολη δήλωση η οποία θα μπορούσε να έχει ξεμείνει από κάποιο ματαιωμένο try block ή να απελευθερώσει οποιονδήποτε εξωτερικό πόρο(όπως χειρισμούς γραφικών, συνδέσεις βάσης δεδομένων ή file streams) χωρίς να περιμένει το συλλογέα απορριμμάτων στο χρόνο εκτέλεσης, για να οριστικοποιήσει τα αντικείμενα. Για παράδειγμα:

```
static void TestFinally()
{
    System.IO.FileStream file = null;
    //Change the path to something that works on your machine
    System.IO.FileInfo fileInfo = new System.IO.FileInfo(@"C:\file.txt");

    try
    {
        file = fileInfo.OpenWrite();
        file.WriteByte(0xF);
    }
    finally
    {
        // Closing the file allows you to reopen it immediately - otherwise IOException is thrown.
        if (file != null)
        {
            file.Close();
        }
    }

    try
    {
```

```
        file = fileInfo.OpenWrite();
        System.Console.WriteLine("OpenWrite() succeeded");
    }
    catch (System.IO.IOException)
    {
        System.Console.WriteLine("OpenWrite() failed");
    }
}
```

Εάν το WriteByte() έριχνε μια εξαίρεση, ο κώδικας στο δεύτερο try block που προσπαθεί να ξανανοίξει το αρχείο θα αποτύγχανε αν το file.Close() δεν είχε κληθεί, και έτσι το αρχείο θα παρέμενε κλειδωμένο. Επειδή τα finally blocks εκτελούνται ακόμα και αν ριχτεί μια εξαίρεση, το finally block του προηγούμενου παραδείγματος επιτρέπει στο αρχείο να κλείσει ορθά και έτσι βοηθά στην αποφυγή ενός σφάλματος.

Generics

Τα Generics χρησιμοποιούνται για να βοηθήσουν να γίνει ο κώδικας, στα συστατικά του λογισμικού, περισσότερο επαναχρησιμοποιήσιμος. Είναι ένας τύπος δομών δεδομένων που περιέχουν κώδικα ο οποίος δεν αλλάζει. Ο τύπος των δεδομένων των παραμέτρων μπορεί να αλλάξει σε κάθε χρήση. Η χρήση μέσα στη δομή των δεδομένων εφαρμόζεται σε διαφορετικούς τύπους των μεταβλητών που έχουν περαστεί.

Κάθε φορά που χρησιμοποιείται ένα Generic, μπορεί να τροποποιηθεί για διαφορετικούς τύπους δεδομένων χωρίς να χρειάζεται να επανεγγραφεί κανένας εσωτερικός κώδικας. Τα Generics επιτρέπουν στις κλάσεις, στις δομές, στις διεπαφές, στις μεθόδους και στα delegates να υποστούν παραμετροποίηση από τους τύπους δεδομένων που αποθηκεύουν και χρησιμοποιούν.

Ένα παράδειγμα όπου θα μπορούσαμε να εφαρμόσουμε generics είναι όταν έχουμε να κάνουμε με συλλογές στοιχείων (integers, strings, orders etc). Μπορούμε να δημιουργήσουμε μια συλλογή από Generics που να μπορεί να χειριστεί οποιονδήποτε τύπο σε ένα generic και σε ένα Type-Safe manner. Για παράδειγμα,

μπορούμε να έχουμε έναν απλό πίνακα κλάσης που να αποθηκεύει μια λίστα χρηστών ή ακόμα και μια λίστα στοιχείων, την οποία όταν την χρησιμοποιήσουμε να έχουμε πρόσβαση στα στοιχεία της συλλογής άμεσα, ως λίστα χρηστών ή στοιχείων και όχι ως λίστα αντικειμένων.

Παρακάτω έχουμε ένα παράδειγμα ορισμού μιας κλάσης generic:

```
// Defining a Generic Class
 class MyArrayList
 {
  private ItemType[] items;
  private int count;
  ...
  public void Add(ItemType item)
  {
   items[count] = item;
  }
  public ItemType GetItem(int index)
  {
   return items[index];
  }
```

Στο παρακάτω παράδειγμα με την χρήση μιας παραμέτρου T μπορούμε να γράψουμε μια ενιαία κλάση την οποία μπορεί να χρησιμοποιήσει άλλος κωδικός client, χωρίς να χρειάζεται να αναλάβουμε το κόστος ή τον κίνδυνο των casts του χρόνου εκτέλεσης ή των λειτουργιών boxing:

```
// Declare the generic class.
public class GenericList<T>
{
   void Add(T input) { }
}
class TestGenericList
{
   private class ExampleClass { }
   static void Main()
   {
      // Declare a list of type int.
      GenericList<int> list1 = new GenericList<int>();

      // Declare a list of type string.
      GenericList<string> list2 = new GenericList<string>();
```

```
        // Declare a list of type ExampleClass.
        GenericList<ExampleClass> list3 = new GenericList<ExampleClass>();
    }
}
```

Οι χρήσεις των generics είναι οι κάτωθι:

- Μεγιστοποιούν την επαναχρησιμοποίηση του κώδικα, την ασφάλεια τύπων και την αποδοτικότητα.
- Η πρότυπη κλάση του .NET Framework περιλαμβάνει μια σειρά από συλλογές κλάσεων στο namespace System.Collections.Generic. Αυτές χρησιμοποιούνται όποτε είναι δυνατό αντί για κλάσεις όπως η ArrayList στο namespace System.Collections.
- Μπορούμε να δημιουργήσουμε δικές μας generic διεπαφές, κλάσεις, μεθόδους, γεγονότα και delegates
- Οι κλάσεις generic μπορούν να περιοριστούν για να επιτρέπουν την πρόσβαση στις μεθόδους σε συγκεκριμένους τύπους στοιχείων
- Οι πληροφορίες των τύπων που χρησιμοποιούνται σε ένα generic τύπο δεδομένων μπορεί να αποκτηθεί στον χρόνο εκτέλεσης με την χρήση της αντανάκλασης.

3.5 Βιβλιοθήκες

3.5.1 Βιβλιοθήκη Πρότυπης Κλάσης

Η Βιβλιοθήκη Πρότυπης Κλάσης (BCL), που είναι τμήμα της Βιβλιοθήκης Κλάσης Framework (FCL), είναι μια βιβλιοθήκη λειτουργικότητας που είναι διαθέσιμη σε όλες τις γλώσσες που χρησιμοποιούν το .NET Framework. Το .NET περιλαμβάνει την BCL ώστε να ενσωματώσει ένα αριθμό κοινών λειτουργιών, συμπεριλαμβανομένων της ανάγνωσης και εγγραφής των αρχείων, της γραφικής

απόδοσης, της αλληλεπίδρασης βάσης δεδομένων, του χειρισμού XML εγγράφων, που κάνουν πιο εύκολη τη δουλειά του προγραμματιστή. Είναι μεγαλύτερη σε έκταση από τις πρότυπες βιβλιοθήκες για τις περισσότερες γλώσσες, συμπεριλαμβανομένης και της C++, και είναι συγκρίσιμη σε έκταση με τις πρότυπες βιβλιοθήκες της Java. Η BCL παρέχει την πιο θεμελιώδη λειτουργικότητα, η οποία περιλαμβάνει κλάσεις σε namespaces όπως: System, System.CodeDom, System.Collections, System.Diagnostics, System.Globalization, System.IO, System.Resources, System.Text, System.Text.RegularExpressions. Η BCL κάποιες φορές αναφέρεται λανθασμένα ως FCL η οποία είναι υπερσύνολο της.

Το .NET εμφανίζεται με μια δική του βιβλιοθήκη κλάσης, τη βιβλιοθήκη Πρότυπης Κλάσης, η οποία παρέχει όλη τη λειτουργικότητα που είναι συσχετισμένη με τις παραδοσιακές βιβλιοθήκες κλάσεων. Είναι ξεχωριστή για δύο κυρίως λόγους:

- Είναι η βιβλιοθήκη κλάσης για την IL και έτσι μπορεί να χρησιμοποιηθεί από οποιαδήποτε γλώσσα που μεταγλωττίζεται σε IL.
- Είναι μια αντικειμενοστραφής βιβλιοθήκη κλάσης και έτσι παρέχει την λειτουργικότητα της μέσω ενός αριθμού κλάσεων που είναι διατεταγμένα σε μια ιεραρχία namespaces

Οι γλώσσες υψηλού επιπέδου παρέχουν τις δικές τους συνδέσεις στις βιβλιοθήκες Πρότυπης Κλάσης, και είναι πιθανό να μην έχουν όλες οι γλώσσες πρόσβαση σε όλα τα χαρακτηριστικά, οι οποίες μπορεί να καταλήξουν να κωδικοποιηθούν στο .NET.

Η βιβλιοθήκη Πρότυπης Κλάσης περιέχει ένα αριθμό συστατικών:

- Ορισμούς βασικών τύπων, όπως Int32. Αυτοί είναι σχεδιασμένοι σε ειδικούς τύπους από ανεξάρτητες γλώσσες.
- Κλάσεις κοινής συλλογής, όπως συστοιχίες, συνδεδεμένες λίστες, πίνακες hash, απαριθμήσεις, ουρές και στοίβες.

- Κλάσεις ορισμού εξαιρέσεων. Όλες οι γλώσσες .NET μπορούν να χρησιμοποιήσουν το χειρισμό εξαίρεσης επειδή είναι ενσωματωμένος στη βιβλιοθήκη Πρότυπη Κλάσης και είναι δυνατό πλέον να δημιουργήσει μια εξαίρεση, ας πούμε, σε μέθοδο γλώσσας C# και να την συλλάβει σε μια αντίστοιχη της γλώσσας VB.
- Κλάσεις για κονσόλες, αρχεία και I/O stream.
- Κλάσεις για προγραμματισμό δικτύων, συμπεριλαμβανομένων και των sockets.
- Κλάσεις διεπαφής βάσεων δεδομένων, συμπεριλαμβανομένων κλάσεων για εργασία με ADO και SQL.
- Κλάσεις γραφικών, συμπεριλαμβανομένου του σχεδιασμού σε 2D, imaging και εκτυπώσεις.
- Κλάσεις για την ανάπτυξη γραφικών διεπαφών χρήστη(GUIs).
- Κλάσεις για σειριοποίηση αντικειμένων.
- Κλάσεις για την εφαρμογή και το χειρισμό πολιτικών ασφάλειας.
- Κλάσεις για την ανάπτυξη κατανεμημένων, συστημάτων βασισμένων στο Web.
- Κλάσεις για εργασία με την XML.
- Άλλα χαρακτηριστικά για τα λειτουργικά συστήματα, όπως ίνες και χρονόμετρα (timers).

Η βιβλιοθήκη Πρότυπης Κλάσης είναι μια αντικειμενοστραφής, μη γλωσσική, συγκεκριμένη αντικατάσταση για τα παλαιότερα Windows API, η οποία παρέχει ένα μεγάλο εύρος υπηρεσιών για το γράψιμο μοντέρνων εφαρμογών, οι οποίες κάνουν μεγάλη χρήση του Web, ανταλλάσσουν δεδομένων και GUIs.

3.5.2 Βιβλιοθήκη Κλάσης Framework

Η βιβλιοθήκη Κλάσης .NET Framework είναι μια μεγάλη συλλογή από περισσότερες από 2500 επαναχρησιμοποιήσιμες κλάσεις, διεπαφές και value types. Η βιβλιοθήκη κλάσης χωρίζεται σε ποικίλλα ιεραρχημένα namespaces με βάση την

λειτουργικότητα και έτσι είναι πιο εύκολο να διαχειριστούν. Το namespace System είναι η ρίζα των namespaces για τους θεμελιώδεις τύπους στο .NET Framework. Η Βιβλιοθήκη Κλάσης .NET Framework περιέχει τις πρότυπες κλάσεις(ενοποιημένα, αντικειμενοστραφή, ιεραρχημένα και επεκτάσιμα σύνολα κλάσεων) που παρέχουν πολλές από τις υπηρεσίες και πολλά από τα αντικείμενα που χρειάζονται για την ανάπτυξη των εφαρμογών. Η Βιβλιοθήκη Κλάσης είναι οργανωμένη σε namespaces. Τα namespaces δεν είναι τίποτα περισσότερο από συσχετισμένες κλάσεις που είναι λογικά ομαδοποιημένες. Για παράδειγμα, το namespace System.Web.Services αποτελείται από κλάσεις που διευκολύνουν την ανάπτυξη και την χρήση των Web Services.

Τα κύρια μέρη της Βιβλιοθήκης Κλάσης.NET Framework είναι τα παρακάτω:

- Πρότυπη Βιβλιοθήκη Κλάσης (ανάμεσα σε άλλα είναι τα System, System.IO, System.Text, System.Reflection, System.Security, System.Net και System.Collections)
- Web Services/Web UI (System.Web)
- Windows UI (System.Windows.Forms)
- Data and XML (System.Data, System.Xml)
- COM+ Services (System.EnterpriseServices)

Παρακάτω αναφέρονται κάποια από τα πιο χρησιμοποιούμενα namespaces:

- Το Microsoft.CSharp περιέχει κλάσεις που υποστηρίζουν την μεταγλώττιση και την παραγωγή κώδικα με την χρήση της γλώσσας C#
- Το System περιέχει θεμελιώδεις κλάσεις και πρότυπες κλάσεις που ασχολούνται με θεμελιακούς τύπους δεδομένων, γεγονότα και χειριστές γεγονότων, διεπαφές, γνωρίσματα κ.ο.κ
- Το System.IO περιέχει κλάσεις για πρότυπη πρόσβαση και διαχείριση ροής δεδομένων, συμπεριλαμβανομένων αρχείων I/O, buffering κ.ο.κ

- Το System.Reflection περιέχει κλάσεις και διεπαφές για να έχει πρόσβαση σε τύπους μεταδεδομένων και για τη δυναμική δημιουργία και επίκληση των τύπων.
- Το System.Runtime.InteropServices περιέχει κλάσεις για τη διαλειτουργικότητα με την COM και με άλλους μη διαχειριζόμενους κώδικες.
- Το System.Runtime.Remoting περιέχει κλάσεις για τη δημιουργία και την τροποποίηση κατανεμημένων εφαρμογών.
- Το System.Net περιέχει κλάσεις συσχετισμένες με τον προγραμματισμό δικτύων (request/response, sockets κ.ο.κ.).
- Το System.Security περιέχει κλάσεις που ασχολούνται με τις άδειες, την κρυπτογραφία κ.ο.κ.
- Το System.Windows.Forms περιέχει κλάσεις για τη δημιουργία παραδοσιακών Windows-based εφαρμογών.
- Το System.Web περιέχει κλασεις για την ανάπτυξη εφαρμογών web, συμπεριλαμβανομένου και του ASP.NET.
- Το System.Web.Services περιέχει κλάσεις που δίνουν τη δυνατότητα ανάπτυξης και χρήσης SOAP-based web services.
- Το System.Data περιέχει κλάσεις για όλες τις λειτουργίες που είναι συσχετισμένες με τη βάση δεδομένων.
- Το System.Data.OleDb περιέχει κλάσεις που υποστηρίζουν τον παροχέα OLE DB .NET.
- Το System.Data.SqlClient περιέχει κλάσεις που υποστηρίζουν τον παροχέα δεδομένων SQL Server .NET.
- Το System.XML περιέχει κλάσεις για τη δημιουργία και επεξεργασία εγγράφων XML.
- Το System.Collections περιέχει κλάσεις για τη συλλογή αντικειμένων όπως ArrayList, Queue και SortedList.
- Το System.Threading περιέχει κλάσεις και διεπαφές που δίνουν τη δυνατότητα για πολυνηματικό προγραμματισμό (Mutex, Thread, and Timeout).

- Το System.Drawing περιέχει κλάσεις για πλούσια λειτουργικότητα γραφικών δύο διαστάσεων και για πρόσβαση στην λειτουργικότητα GDI+.
- Το System.Drawing.Drawing2D παρέχει προχωρημένη λειτουργικότητα δυσδιάστατων και διανυσματικών (vector) γραφικών.
- Το System.Drawing.Imaging παρέχει προχωρημένη GDI+ imaging λειτουργικότητα.
- Το System.Drawing.Printing περιέχει κλάσεις που επιτρέπουν την τροποποίηση στην εκτύπωση.
- Το System.Drawing.Text παρέχει προχωρημένη τυπογραφική λειτουργικότητα GDI+

3.6 Διαχείριση μνήμης

3.6.1 Στοίβα και σωρός

Στην επιστήμη των υπολογιστών η στοίβα είναι ένας ειδικός τύπος δομής δεδομένων στην οποία τα στοιχεία αφαιρούνται με την αντίστροφη σειρά από αυτή που προστίθενται, έτσι το στοιχείο εκείνο που προστέθηκε πρώτο είναι αυτό που θα αφαιρεθεί τελευταίο. Η μέθοδος αυτή ονομάζεται LIFO (last in first out). Η στοίβα μπορεί να έχει οποιονδήποτε αφαιρούμενο τύπο δεδομένων ως στοιχείο, αλλά χαρακτηρίζεται από μόνο δύο θεμελιώδεις λειτουργίες: ώθηση και απώθηση. Η λειτουργία της ώθησης προσθέτει στην κορυφή της λίστας ένα στοιχείο ή αρχικοποιεί την στοίβα σε περίπτωση που αυτή είναι άδεια. Η λειτουργία της απώθησης αφαιρεί ένα στοιχείο από την κορυφή της λίστας και επιστρέφει την τιμή της σε αυτόν που έκανε την κλήση.

Η στοίβα είναι μια περιορισμένη δομή δεδομένων επειδή μόνο ένας μικρός αριθμός λειτουργιών μπορούν να εκτελεστούν σε αυτήν. Η φύση των λειτουργιών ώθησης και απώθησης έχουν ως συνέπεια να είναι η σειρά των στοιχειών της στοίβας φυσιολογική. Επειδή τα στοιχεία της στοίβας αφαιρούνται με τον αντίστροφο τρόπο από τον οποίο εισάγονται αυτό έχει ως συνέπεια το στοιχείο που εισήχθη πρώτο να παραμένει περισσότερο μέσα στη στοίβα.

Παρακάτω δίνονται παραδείγματα δημιουργίας της στοίβας καθώς και των λειτουργιών ώθησης και απώθησης:

```
typedef struct {
  int size;
  int items[STACKSIZE];
} STACK;
```

Η λειτουργία ώθησης χρησιμοποιείται για να αρχικοποιήσει την στοίβα και να αποθηκεύει τιμές σε αυτή. Είναι υπεύθυνη για την εισαγωγή των στοιχείων στον πίνακα ps->items [] και για την αύξηση του στοιχείου μέτρησης (ps->size). Είναι επίσης αναγκαίο να ελέγχεται η στοίβα από την συγκεκριμένη λειτουργία για να αποτραπεί η περίπτωση υπερχείλισης.

```
void push(STACK *ps, int x)
  {
    if (ps->size == STACKSIZE) {
      fputs("Error: stack overflow\n", stderr);
      abort();
    } else
      ps->items[ps->size++] = x;
  }
```

Η λειτουργία απώθησης είναι υπεύθυνη για την αφαίρεση ενός στοιχείου από την στοίβα και για να μειώνει την τιμή του ps->size. Επίσης θα πρέπει να διενεργεί έλεγχο έτσι ώστε να αποτρέπεται η περίπτωση υποχείλισης.

```
int pop(STACK *ps)
  {
    if (ps->size == 0){
      fputs("Error: stack underflow\n", stderr);
      abort();
    } else
      return ps->items[--ps->size];
    }
```

3.6.2 Σύγκριση στοίβας και σωρού

Η στοίβα είναι περισσότερο υπεύθυνη για να ελέγχει τι εκτελείται στον κώδικα (ή τι καλείται). Ο σωρός είναι υπεύθυνος για τον εντοπισμό των αντικειμένων.

Ο σωρός μοιάζει με την στοίβα ο σκοπός του όμως είναι να κρατά πληροφορίες, έτσι ώστε οτιδήποτε μέσα στο σωρό να είναι προσβάσιμο οποιαδήποτε στιγμή. Με τον σωρό δεν υπάρχει περιορισμός στο τι είναι προσβάσιμο όπως στη στοίβα. Ο σωρός είναι σαν το σωρό των καθαρών ρούχων επάνω σε ένα κρεβάτι από το οποίο μπορούμε να πάρουμε όποιο ρούχο θέλουμε.

Η στοίβα είναι αυτό-διατηρούμενη από την άποψη ότι ελέγχει τη δική της διαχείριση μνήμης. Όταν το ανώτερο στοιχείο της στοίβας δεν χρησιμοποιείται τότε απορρίπτεται. Αντίθετα, ο σωρός ασχολείται με την συλλογή απορριμμάτων, η οποία φροντίζει ώστε να μένει καθαρός ο σωρός.

Υπάρχουν 4 κύριοι τύποι στοιχείων που τοποθετούνται στις στοίβες και στους σωρούς κατά τη διάρκεια της εκτέλεσης του κώδικα: Value Types, Reference Types, Pointers, and Instructions.

Έχοντας αναφερθεί σε προηγούμενη ενότητα στα Value και Reference Types σε προηγούμενη ενότητα, θα κάνουμε αναφορά μόνο στους Poniters και στα Instructions.

Pointers: ο τρίτος τύπος στοιχείου που μπαίνει στο πρόγραμμα διαχείρισης μνήμης μας είναι μια Reference σε ένα τύπο. Η Reference συχνά αναφέρεται και ως Pointer. Δεν χρησιμοποιούμε αυστηρά Pointers που διαχειρίζονται από το CLR. Ένας Pointer είναι διαφορετικός από ένα Reference Type, καθώς όταν

αναφερόμαστε σε Reference Type σημαίνει πως αυτό είναι προσβάσιμο μέσω ενός Pointer. Ένας Pointer είναι ένα τύπος δεδομένων στην μνήμη που παραπέμπει σε άλλο τύπο στην μνήμη. Ένας Pointer πιάνει χώρο στην μνήμη σαν οτιδήποτε άλλο αντικείμενο που τοποθετούμε στην στοίβα ή στον σωρό και η τιμή του είναι ή μια διεύθυνση μνήμης ή μηδενική.

Instructions: μια instruction είναι μια ενιαία λειτουργία ενός επεξεργαστή που έχει καθοριστεί από ένα σύνολο αρχιτεκτονικής instruction. Με την ευρύτερη έννοια, μια instruction μπορεί να είναι μια αναπαράσταση ενός στοιχείου του εκτελέσιμου προγράμματος όπως το bytecode.

Μια instruction περιλαμβάνει ένα opcode που προσδιορίζει την εκτέλεση της λειτουργίας, όπως «πρόσθεσε συστατικά μνήμης στο μητρώο», και κανένα ή περισσότερα προσδιοριστικά τελεσταίων, τα οποία μπορούν να προσδιορίσουν εγγραφές, τοποθεσίες μνήμης ή literal δεδομένα. Τα προσδιοριστικά τελεσταίων μπορεί να έχουν addressing modes, τα οποία καθορίζουν την σημασία των προσδιοριστικών.

Υπάρχουν δύο χρυσοί κανόνες:

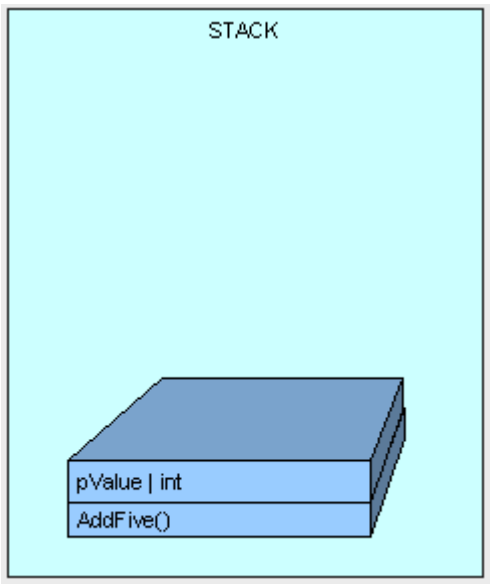
1. ένας Reference Type πάει πάντα στον σωρό

2. οι Value Types και οι Pointers πάνε πάντα εκεί που δηλώνονται. Είναι όμως λίγο περίπλοκο να βρει η στοίβα που έχουν δηλωθεί που έχουν δηλωθεί τα «αντικείμενα».

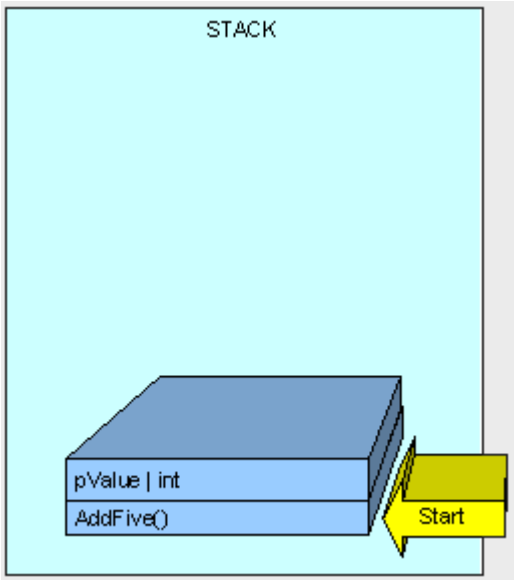
Η στοίβα είναι υπεύθυνη για να ελέγχει που βρίσκεται το κάθε νήμα κατά τη διάρκεια της εκτέλεσης του κώδικα. Κάθε νήμα έχει τη δική του στοίβα. Όταν ο κώδικας διενεργεί μια κλήση για να εκτελεστεί η μέθοδος, το νήμα ξεκινά να εκτελεί τις instructions που έχουν μεταγλωττιστεί JIT και βρίσκονται στον πίνακα μεθόδου και τοποθετεί τις παραμέτρους της μεθόδου στην στοίβα νήματος.

Για παράδειγμα, η ακόλουθη μέθοδος:

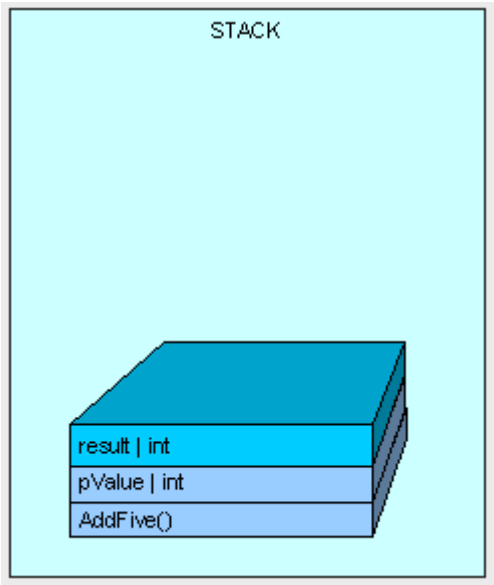
```
public int AddFive(int pValue)
    {
        int result;
        result = pValue + 5;
        return result;
    }
```



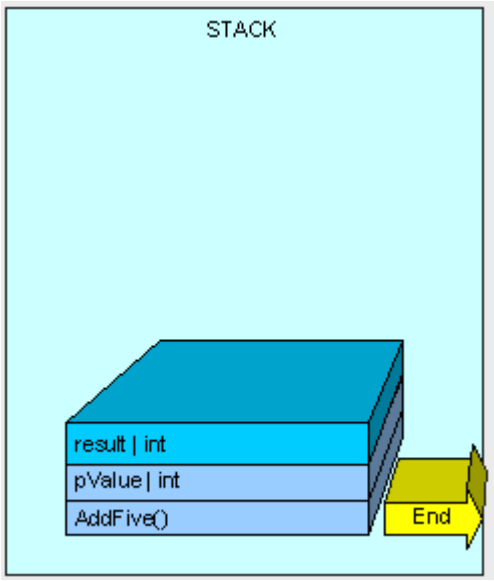
Καθώς εκτελείται η μέθοδος, οι παράμετροι της τοποθετούνται στην στοίβα. Ο έλεγχος μετά περνάει στις instructions της μεθόδου AddFive, η οποία βρίσκεται στον πίνακα μεθόδου των τύπων. Εκτελείται μια μεταγλώττιση JIT εάν είναι η πρώτη φορά που εκτελείται η μέθοδος.



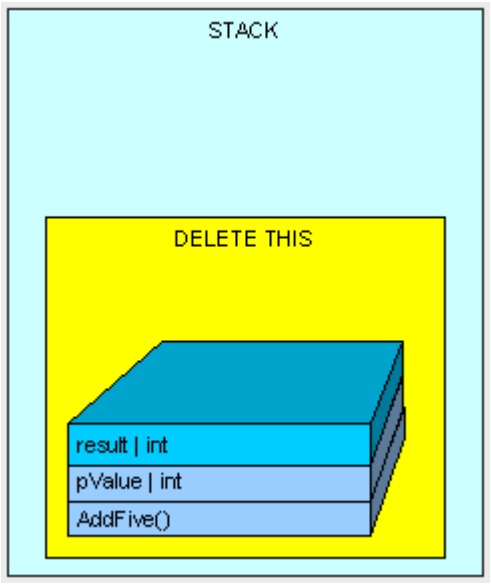
Καθώς εκτελείται η μέθοδος, χρειαζόμαστε χώρο στην μνήμη για την μεταβλητή “result” η οποία κατανέμεται στην στοίβα.



Τερματίζεται η εκτέλεση της μεθόδου και το αποτέλεσμα μας, επιστρέφεται.



Τώρα όλη η κατανεμημένη μνήμη στη στοίβα καθαρίζεται με την μετακίνηση ενός pointer στη διεύθυνση της διαθέσιμης μνήμης όπου ξεκίνησε η AddFive( ) και επιστρέφουμε στην προηγούμενη μέθοδο της στοίβας (δεν φαίνεται στο παρακάτω σχήμα).



Σε αυτό το σχήμα η μεταβλητή result τοποθετείται στην στοίβα. Κάθε φορά που δηλώνεται ένας Value Type μέσα στην μέθοδο, τοποθετείται στην στοίβα. Οι Value Types τοποθετούνται κάποιες φορές στο σωρό. Αν ο Value Type δηλωθεί έξω από την μέθοδο αλλά μέσα σε ένα Reference Type, θα τοποθετηθεί μέσα στο Reference Type στον σωρό.

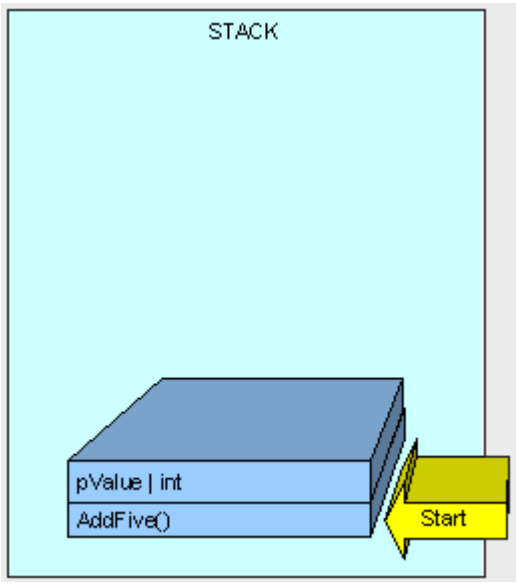
Για παράδειγμα: αν έχουμε την ακόλουθη κλάση:

```
public class MyInt
   {
     public int MyValue;
   }
```

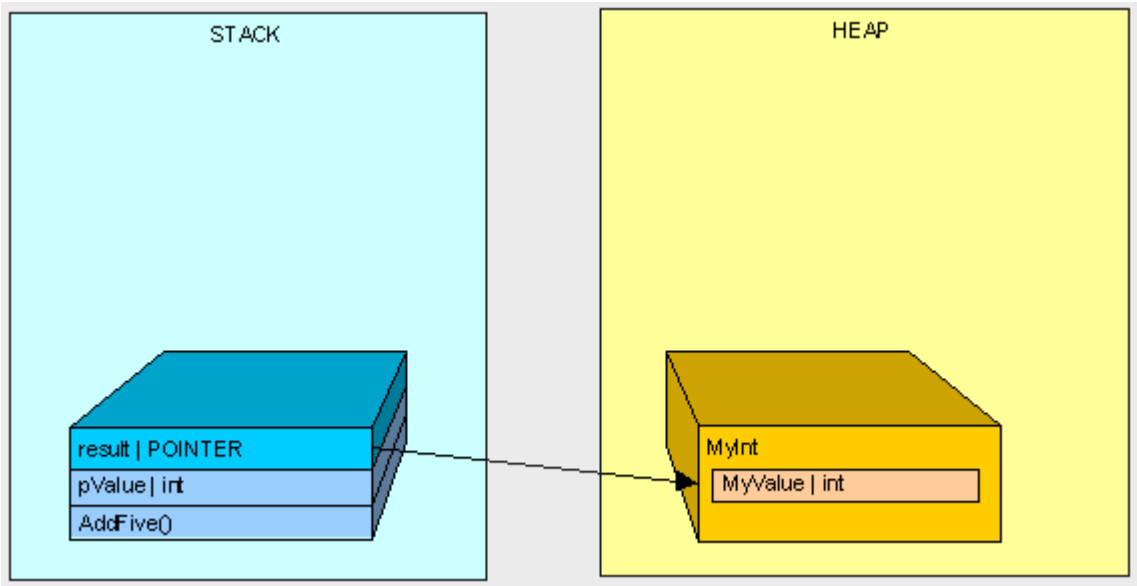
και εκτελείται η ακόλουθη μέθοδος:

```
public MyInt AddFive(int pValue)
   {
        MyInt result = new MyInt();
        result.MyValue = pValue + 5;
        return result;
   }
```

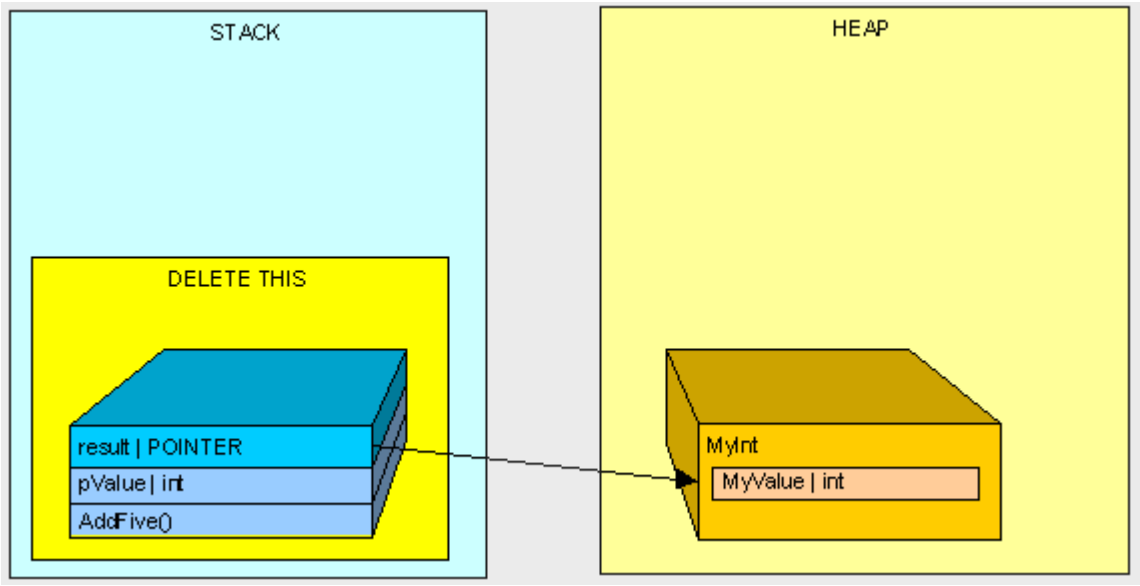
Το νήμα θα ξεκινήσει την εκτέλεση της μεθόδου και οι παράμετροι της θα τοποθετηθούν στην στοίβα του νήματος.



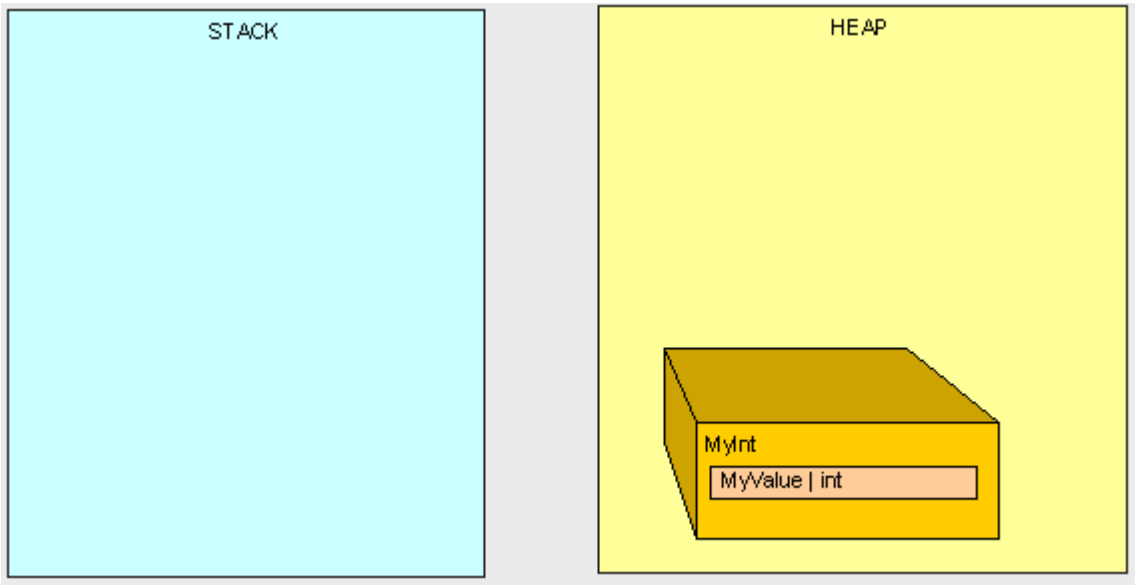
επειδή η MyInt είναι Reference Type τοποθετείται στον σωρό και αναφέρεται (referenced) από ένα Pointer στην στοίβα.



αφού η AddFive( ) ολοκληρώσει την εκτέλεση (όπως στο πρώτο παράδειγμα), γίνεται ο καθαρισμός…



μας έχει ξεμείνει μια MyInt στον σωρό(δεν υπάρχει τίποτα στην στοίβα που να δείχνει την Myint)



Όταν φτάσουμε σε ένα συγκεκριμένο σημείο στην μνήμη και χρειαζόμαστε περισσότερο χώρο στον σωρό, τότε ξεκινάει ο συλλογέας απορριμμάτων. Ο συλλογέας σταματάει όλα τα νήματα που εκτελούνται, βρίσκει τα αντικείμενα που δεν χρησιμοποιούνται στον σωρό και τα αφαιρεί. Ο συλλογέας αναδιοργανώνει τα αντικείμενα που απέμειναν στον σωρό για να δημιουργηθεί χώρος και αντιστοιχεί τους Pointers στα αντικείμενα που βρίσκονται και στο σωρό και στην στοίβα.

Εάν εκτελέσουμε την παρακάτω μέθοδο:

```
public int ReturnValue()
 {
     int x = new int();
     x = 3;
     int y = new int();
     y = x;
     y = 4;
     return x;
 }
```

Θα έχουμε ως αποτέλεσμα την τιμή 3. εάν χρησιμοποιήσουμε την κλάση MyInt όπως σε προηγούμενο παράδειγμα:

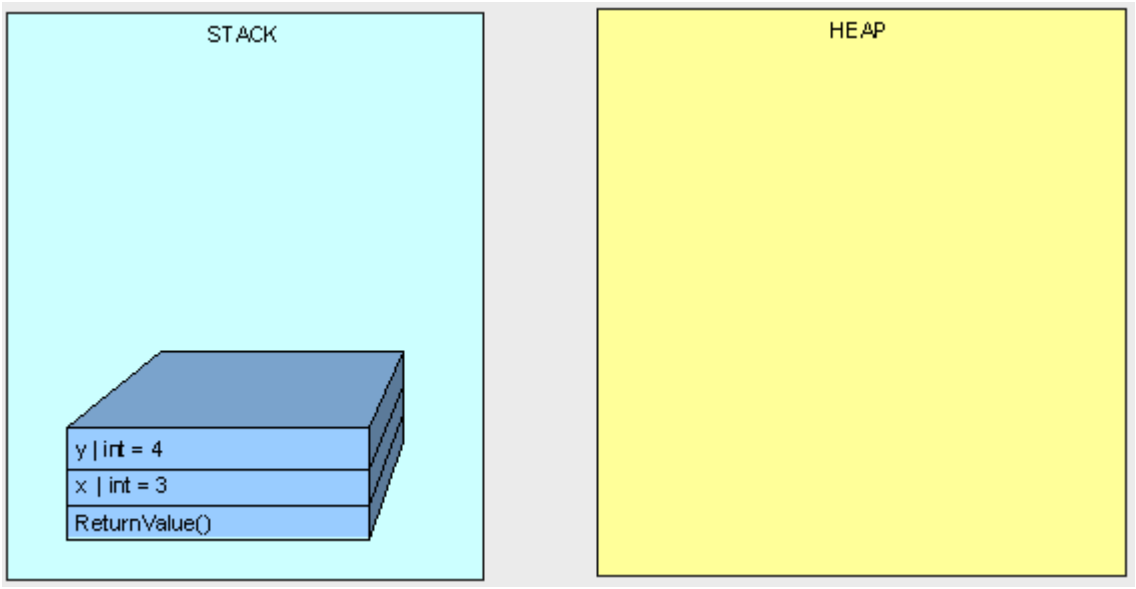
```
public class MyInt
    {
        public int MyValue;
    }
```

Θα εκτελεστεί η παρακάτω μέθοδος:

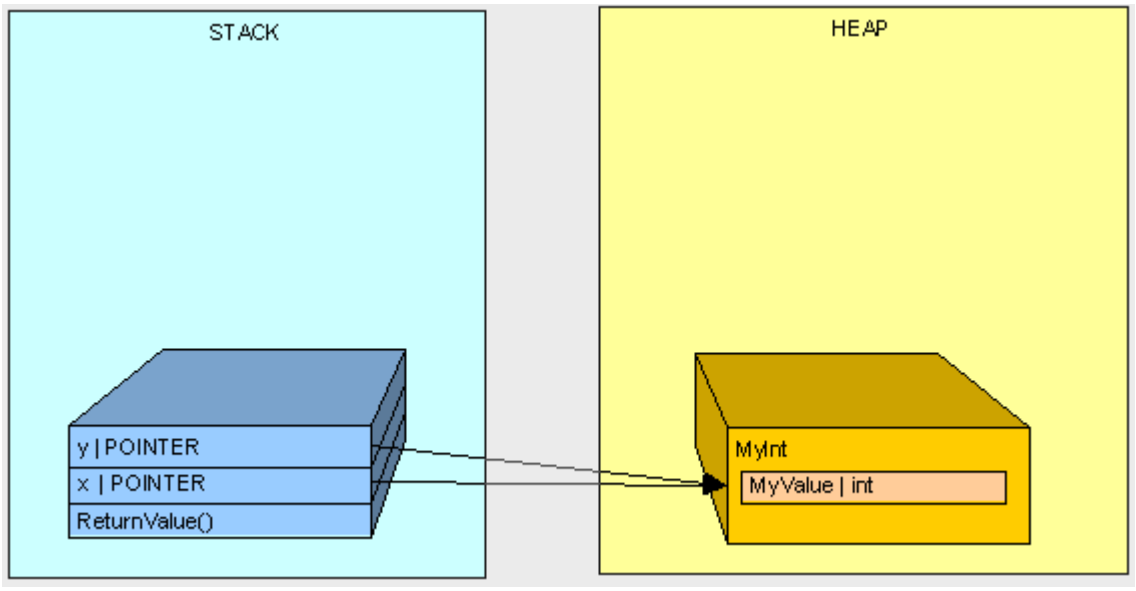
```
public int ReturnValue2()
    {
        MyInt x = new MyInt();
        x.MyValue = 3;
        MyInt y = new MyInt();
        y = x;
        y.MyValue = 4;
        return x.MyValue;
    }
```

Εδώ λαμβάνουμε την τιμή 4. Αυτό συμβαίνει γιατί σε αντίθεση με το προηγούμενο παράδειγμα γιατί οι μεταβλητές x,y δείχνουν το ίδιο αντικείμενο στον σωρό. Αυτό φαίνεται διαγραμματικά και παρακάτω:

```
public int ReturnValue()
    {
        int x = 3;
        int y = x;
        y = 4;
        return x;
    }
```




```
public int ReturnValue2()
        {
            MyInt x;
            x.MyValue = 3;
            MyInt y;
            y = x;
            y.MyValue = 4;
            return x.MyValue;
        }
```



3.6.3 Συλλογή απορριμμάτων

Στην επιστήμη των υπολογιστών η συλλογή απορριμμάτων είναι μια μορφή αυτόματης διαχείρισης μνήμης. Είναι μια ειδική περίπτωση διαχείρισης πόρων, στην οποία οι περιορισμένοι πόροι οι οποίοι διαχειρίζονται είναι η μνήμη. Ο συλλογέας απορριμμάτων δοκιμάζει να ανακτήσει τα απορρίμματα ή την μνήμη που είναι κατειλλημένη από αντικείμενα που δεν χρησιμοποιούνται πλέον από το πρόγραμμα.

Η συλλογή απορριμμάτων παραδοσιακά δε διαχειρίζεται άλλους περιορισμένους πόρους παρά την μνήμη που χρησιμοποιούν τα κοινά προγράμματα, όπως τα sockets δικτύων, τους χειρισμούς βάσεων δεδομένων, τα παράθυρα αλληλεπίδρασης χρηστών και τους περιγραφείς αρχείων και συσκευών. Κάποια συστήματα συλλογής απορριμμάτων επιτρέπουν σε πόρους σαν και αυτούς να συσχετιστούν με μια περιοχή μνήμης, έτσι ώστε όταν συλλεχθούν, να αναγκαστούν οι άλλοι πόροι να επανακτηθούν. Αυτή η διαδικασία λέγεται οριστικοποίηση.

Στη γλώσσα προγραμματισμού C#, η απελευθέρωση των πόρων που χρησιμοποιήθηκαν από τις δημιουργημένες instances συμβαίνει αυτόματα, σε ένα πρόγραμμα καθορισμένο από το σύστημα, από ένα μηχανισμό γνωστό ως συλλογή απορριμμάτων. Ο συλλογέας απορριμμάτων είναι σαν ένα ψάρι σκούπα σε ενυδρείο. Εάν ρίξουμε περισσότερο φαγητό από όσο μπορούν να φάνε τα άλλα ψάρια, το ψάρι σκούπα καταναλώνει ο,τι έχει απομείνει χωρίς να αφήσει τίποτα πίσω του.

Η άνεση και η χρησιμότητα του συλλογέα απορριμμάτων είχε διαπιστωθεί μέσω της Java. Ένας παρόμοιος μηχανισμός εισήχθη και στην C++.

Ο συλλογέας απορριμμάτων στη C# βελτιώνει την ιδέα επειδή δεν μειώνει την επίδοση του συστήματος, μέχρι αυτό να είναι πραγματικά απαραίτητο. Είναι πιο ευφυής από οποιονδήποτε άλλο συλλογέα απορριμμάτων στην αγορά. Μπορούν να ξεπεραστούν τα μειονεκτήματα της μη καθορισμένης συμπεριφοράς του, με την χρήση destructors και finalizers καθώς και να κατανεμηθούν τα instances των μελών με τους constructors.

Στη C#, τις περισσότερες φορές, δε χρειάζεται να κωδικοποιηθούν οι destructors ή οι finalizers επειδή καθαρίζει τα υπολείμματα για εμάς ο συλλογέας απορριμμάτων. Αυτές οι συναρτήσεις καλούνται ρητά από τον χρόνο εκτέλεσης του συστήματος, του συλλογέα απορριμμάτων.

Εάν η ρητή απελευθέρωση των αντικειμένων είναι σημαντική στις κλάσεις και στην ιεραρχία της κληρονομικότητας, πρέπει να βεβαιωθούμε ότι οι κλάσεις κληρονομούν τη διεπαφή IDisposable και εφαρμόζουν τη συνάρτηση Dispose. Αυτή η συνάρτηση μπορεί να κληθεί οποιαδήποτε στιγμή θέλουμε να οριστικοποιήσουμε ένα αντικείμενο και να απελευθερώσουμε τους πόρους.

Στο παρακάτω παράδειγμα καταδεικνύεται η έννοια της συλλογής απορριμμάτων στη C#:

```
  class X
{
   // implicit dtor ~X()
   // created for you automatically by C#
}

class Y
{
   ~Y() // explicit dtor, same as Finalize
   {
      // some code
   }
}

//verbose syntax:

class Z
{
   protected override void Finalize() //verbose explicit dtor, same as ~
```

```
    {
        // Some code
        // implicit call to base.Finalize();
    }
}
```

Εάν παρατηρήσουμε, στην ενδιάμεση γλώσσα που έχει παραχθεί για κάθε οδηγία του .NET Framework μέσω του ildasm.exe, βλέπουμε ότι ο μεταγλωττιστής μεταδίδει το όνομα ~Y ως Finalize. Οι Finalizers της C# δεν καλούν πραγματικά άλλους Finalizers, συμπεριλαμβανομένων και αυτών της πρότυπης κλάσης. Εντούτοις, συχνά θέλουμε να οριστικοποιήσουμε και ένα παραγόμενο αντικείμενο και την πρότυπη κλάση του στην ίδια κλήση. Επιπλέον, κάθε κλάση C# που γράφουμε έχει μόνο μια πρότυπη κλάση, ακόμα και αν αυτή είναι σιωπηλά System.Object.

Όταν βγαίνουμε από το πρόγραμμα, μπορούμε να κάνουμε τις μεθόδους System.GC να εξαναγκάσουν τον συλλογέα απορριμμάτων να προγραμματίσει τη ρητή κλήση finalizer, και ύστερα να περιμένουμε να ολοκληρωθεί, όπως κάτωθι:

```
    System.GC.Collect();
System.GC.WaitForPendingFinalizers();
```

Μπορούμε επίσης να καλέσουμε την μέθοδο System.GC.SuppressFinalize για να αποτραπεί, ο συλλογέας απορριμμάτων, από την σιωπηλή επίκληση του Finalizer, για δεύτερη φορά.

Η μέθοδος Finalize καλείται αυτόματα από τον destructor. Οι destructors και το object.Finalize δεν μπορούν να κληθούν άμεσα. Πρέπει να κληθεί το IDisposable.Dispose, εάν είναι διαθέσιμο.

Για να καταστραφεί ρητά ένα αντικείμενο, πρέπει να κληθεί άμεσα η System.GC.SuppressFinalize για να αποτραπεί η πολλαπλή απελευθέρωση αντικειμένων από το συλλογέα απορριμμάτων:

```
System.GC.SuppressFinalize(anyInstance);
```

Εάν έχει σημασία η σειρά με την οποία θα καταστραφούν τα αντικείμενα, πρέπει να αποθηκευθεί σε μια στοίβα η σειρά με την οποία δημιουργήθηκαν τα αντικείμενα για τα οποία θα κληθεί η οριστικοποίηση.

Κάτωθι δίνεται παράδειγμα με τους Destructors:

```
// example: explicit destruction/finalize
using System;

class MyClass : IDisposable
{
   public MyClass() //default ctor
   {
      this.iNumber = 0;
      System.Console.WriteLine("ctor:MyClass {0}", iNumber);
   }

   public MyClass(Int32 iNumber) // specialized ctor
   {
      this.iNumber = iNumber;
      System.Console.WriteLine("ctor:MyClass {0}", iNumber);
   }

   ~MyClass() // dtor or finalize
   {
      System.Console.WriteLine("dtor:~MyClass {0}", iNumber);
   }

   public void Dispose() // helper finalize function
   {
```

```
        // here you can free the resources you allocated explicitly
        System.GC.SuppressFinalize(this);
    }

    private int iNumber;
}

class main
{
    static void Main()
    {
        MyClass myClass1 = new MyClass();
        MyClass myClass2 = new MyClass(19);
        myClass1.Dispose(); // myClass1 is explicitly exposed.
        System.GC.Collect();
        System.GC.WaitForPendingFinalizers();

        // myClass2 is implicitly exposed by GC.
        Console.ReadLine();
    }
}
```

Οι constructors και οι destructors έχουν συγκεκριμένη σειρά εκτέλεσης για τις συνδεδεμένες, εξαγόμενες κλάσεις. Αυτή η σειρά είναι σημαντική για την απελευθέρωση των πόρων. Ας υποθέσουμε ότι υπάρχει η ακόλουθη απλή αρχιτεκτονική κλάσης:

```
    class A{} // base
classB:A{} // B inherits A
class C:B{} // C inherits B
```

Όταν δημιουργείται ένα C αντικείμενο με τις ακόλουθες προδιαγραφές, οι constructors καλούνται με την ακόλουθη σειρά: πρώτα ο A, μετά ο B, και μετά ο C.

```
C c = new C();
```

Οι destructors θα κληθούν με την αντίστροφη σειρά: πρώτα ο C, μετά ο B και μετά ο A.

Από την στιγμή που ο συλλογέας απορριμμάτων αναλαμβάνει την απελευθέρωση, η σειρά καταστροφής των αντικειμένων γίνεται από αυτόν. Σε αυτή την περίπτωση τα αντικείμενα μπορεί να καταστραφούν με τυχαία σειρά. Αυτή η συμπεριφορά μπορεί να δημιουργήσει προβλήματα καθώς μπορεί να απελευθερωθούν πόροι στις κλάσεις-γονείς ακόμα και αν χρειάζονται στις εξαγόμενες κλάσεις.

Επίλογος

Σε αυτή την πτυχιακή εργασία προσπαθήσαμε να αναπτύξουμε και να περιγράψουμε την αρχιτεκτονική του .NET Framework, να συγκρίνουμε την πλατφόρμα του .NET με αυτές τις Java, κατά κύριο λόγο όμως να παρουσιάσουμε και να περιγράψουμε την γλώσσα προγραμματισμού C# και τις ευκολίες που παρέχει αυτή στους χρήστες καθώς και τις λειτουργίες και τα χαρακτηριστικά της.

Μέσα από αυτή την πτυχιακή εργασία ο αναγνώστης εξοικειώνεται με τους όρους και τις έννοιες του .NET και του δίνεται η δυνατότητα να κατανοήσει την αρχιτεκτονική του και το που υπερτερεί σε σχέση με τις άλλες πλατφόρμες της Java καθώς και τις ομοιότητες και διαφορές τους.

Επιπροσθέτως, αναλύει σε βάθος τα γενικά στοιχεία της C#, την έννοια της κληρονομικότητας, τα προχωρημένα χαρακτηριστικά της τις βιβλιοθήκες της και τη διαχείριση μνήμης. Με την ανάλυση των παραπάνω δίνεται στον αναγνώστη η δυνατότητα να έχει μια ευρεία και εις βάθος γνώση για την αντικειμενοστραφή γλώσσα C# .

Συνολικά, με την παραπάνω πτυχιακή εργασία δίνεται η δυνατότητα στον αναγνώστη να εισχωρήσει στις βασικές έννοιες της C# και να κατανοήσει αυτές και τον τρόπο λειτουργίας της αντικειμενοστραφούς γλώσσας καθώς και να έρθει σε επαφή με την αρχιτεκτονική του .NET.

Βιβλιογραφία

1. Wikipedia, the free encyclopedia,
   http://en.wikipedia.org/wiki/Main_Page

2. Ι.-Χ. Παναγιωτόπουλος, Από την Java στη C#, Πειραιάς 2003

3. C-Sharp Corner,C# Heap(ing) Vs Stack(ing) in .NET: Part I,
   http://www.c-sharpcorner.com/

4. Devhood,
   http://www.devhood.com/

5. MSDN
   http://msdn.microsoft.com/el-gr/default.aspx

6. CsharpFriends.com, Generics in C#
   http://www.csharpfriends.com/

7. Techotopia, The C# Language& Environment,
   http://www.techotopia.com/index.php/The_C_Sharp_Language_and_Environment#The_Framework_.28Base_Class_and_Framework_Class_Libraries.29