Εισαγωγή στη C# και το .NET 4.0

Σημειώσεις Σεμιναρίου

Επιμέλεια: Βασίλης Κόλιας

Ενότητα 3

- **Κλάσεις και αντικείμενα**
- **Πεδία**
- **Μέθοδοι**

1.0.0

Πίνακας Περιεχομένων

4. Κλάσεις και αντικείμενα

Σε αυτήν την ενότητα γίνεται μια εισαγωγή στον αντικειμενοστρεφή προγραμματισμό έτσι όπως είναι υλοποιημένος στη C#. Η ενότητα αυτή και η επόμενη ενδεχομένως είναι οι πιο σημαντικές καθώς, εξαιτίας της αντικειμενοστρέφειας της C#, δεν νοείται προγραμματιστής που επιθυμεί να τη χρησιμοποιήσει για την ανάπτυξη εφαρμογών, να μην είναι εξοικειωμένος με αυτές.

Όλα τα προγράμματα σε C# χρησιμοποιούν αντικείμενα και ο τύπος των αντικειμένων ορίζεται από την κλάση (class) ή το interface (διασύνδεση) τους. Κάθε πρόγραμμα σε C# ορίζεται με μια κλάση και τα σύνθετα προγράμματα συνήθως περιλαμβάνουν αρκετούς ορισμούς κλάσεων. Ξεκινούμε με μερικές έννοιες:

Μια **κλάση** είναι μια δόμηση που χρησιμοποιείται σαν ένα προσχέδιο για τη δημιουργία στιγμιότυπων του εαυτού της που καλούνται στιγμιότυπα κλάσεων, αντικείμενα κλάσεων ή απλώς αντικείμενα. Μια κλάση ορίζει τα μέλη που την αποτελούν τα οποία δίνουν τη δυνατότητα στα αντικείμενά της να έχουν κατάσταση και συμπεριφορά. Τα πεδία μιας κλάσης διατηρούν την κατάσταση κάθε αντικειμένου της, ενώ οι μέθοδοι (κατά κύριο λόγο) καθορίζουν τη συμπεριφορά των αντικειμένων της.

Όπως αναφέρθηκε στην πρώτη ενότητα, η κλάση αποτελεί τη θεμελιώδη μονάδα του αντικειμενοστρεφούς προγραμματισμού (μαζί με το αντικείμενο), μέσω της οποίας ο προγραμματιστής ορίζει νέους τύπους δεδομένων. Στη C# δεν γίνεται να γραφτεί κώδικας χωρίς να οριστεί μια κλάση και όλες οι μέθοδοι ορίζονται μέσα σε κλάσεις. Με μια κλάση ορίζεται ένας τύπος αναφοράς (reference type). Ορίζοντας λοιπόν μια κλάση ουσιαστικά ορίζουμε έναν νέο τύπο δεδομένων που περιέχει μεταβλητές και μεθόδους.

Μια κλάση μπορεί να θεωρηθεί σαν μια ομαδοποίηση δεδομένων και μεθόδων που λογικά ενώνονται μαζί σε ένα πακέτο. Η C# δίνει μεγάλη ευελιξία στη δημιουργία κλάσεων ωστόσο καλές κλάσεις θεωρούνται αυτές που αναπαριστούν έννοιες του πραγματικού κόσμου. Η Επιστήμη των Υπολογιστών μοντελοποιεί τον κόσμο μέσα από δομές που αναπαριστούν έννοιες ή πράγματα του πραγματικού κόσμου, όπως για παράδειγμα τραπεζικούς λογαριασμούς, πελάτες, έγγραφα ή προϊόντα. Για τους αναλυτές προγραμμάτων οι κλάσεις αντικατοπτρίζουν τις έννοιες ενός προβλήματος σε ένα πρόγραμμα. Για παράδειγμα ένα πρόβλημα θα μπορούσε να είναι η στατιστική ανάλυση των επιδόσεων των φοιτητών μιας σχολής για την βελτίωση του οδηγού σπουδών. Οποιαδήποτε περιγραφή του προβλήματος αυτού θα περιείχε τον όρο φοιτητής σε κάθε πιθανή λύση του. Οι φοιτητές έχουν μερικές ιδιότητες που πρέπει να ληφθούν υπόψη σε μια τέτοια ανάλυση, όπως ο μέσος όρος των μαθημάτων όπου έχουν επιτύχει, τα χρωστούμενα μαθήματα, οι απουσίες τους, η ηλικία τους. Επιπρόσθετα οι φοιτητές εκδηλώνουν κάποιες συμπεριφορές, όπως για παράδειγμα ότι δηλώνουν τα μαθήματα που τους ενδιαφέρουν σε κάθε εξάμηνο, δίνουν εξετάσεις, παρακολουθούν μαθήματα, δίνουν ασκήσεις κ.α. Επομένως η έννοια του φοιτητή είναι μέρος του προβλήματος που περιγράφτηκε και κάθε καλό πρόγραμμα που θέλει να κάνει μια τέτοια ανάλυση θα πρέπει να εκφράσει κάπως την έννοια του φοιτητή. Αυτό στη C# (και κάθε άλλη αντικειμενοστρεφή γλώσσα) γίνεται με τη χρήση μιας κλάσης Student.

Ένα **αντικείμενο** από την άλλη είναι ένα στιγμιότυπο μιας κλάσης. Αν η έννοια του φοιτητή αναπαρίσταται με μια κλάση, τότε ένας πραγματικός φοιτητής αναπαρίσταται με ένα αντικείμενο. Ένα από τα ζητούμενα στην ανάπτυξη αντικειμενοστρεφών προγραμμάτων είναι το πώς θα αναπαρασταθούν αντικείμενα του πραγματικού κόσμου μέσα στο πρόγραμμα, μέσω των κλάσεων. Ένας απλούστερος ορισμός λοιπόν θα ήταν ότι το αντικείμενο αποτελεί ένα επαναχρησιμοποιήσιμο κομμάτι λογισμικού που αναπαριστά ένα αντικείμενο του πραγματικού κόσμου.

4.1. Δήλωση κλάσης

Όπως αναφέραμε, μια κλάση είναι το πιο συνηθισμένο είδος τύπου αναφοράς. Η πιο απλή δήλωση κλάσης είναι η ακόλουθη:

```
class όνομα-κλάσης
{
}
```

Μια πιο πολύπλοκη δήλωση κλάσης θα περιελάμβανε τα ακόλουθα:

Πριν τη λέξη class	• Ένα σύνολο *χαρακτηριστικών (attributes)* • Ένα σύνολο *τροποποιητών (modifiers)*. Αυτοί είναι οι public, internal, abstract, sealed, static, unsafe και partial
Μετά το όνομα-κλάσης	• Μια *λίστα παραμέτρων τύπων (type parameter list)* • Μια κλάση γονέα (base class) • Ένα ή περισσότερα interfaces
Μεταξύ των αγκίστρων	• Ένα σύνολο μελών κλάσης. Αυτά μπορεί να είναι μέθοδοι (methods), ιδιότητες (properties), indexers, πεδία (fields), κατασκευαστές (constructors), operator functions, φωλιασμένοι τύποι και ένας καταστροφέας.

Στην απλοϊκή δήλωση της κλάσης παραπάνω, μετά τη δεσμευμένη λέξη class ακολουθεί ένας προσδιοριστής. Όπως σε όλους τους προσδιοριστές, ισχύουν και εδώ οι κανόνες ονοματοδοσίας (παράγραφος 3.1.1), ενώ υπάρχει και η σύμβαση να ακολουθείται το στυλ γραφής της Pascal, δηλαδή το πρώτο γράμμα του προσδιοριστή να είναι κεφαλαίο, όπως και τα πρώτα γράμματα των συνενωμένων λέξεων (αν υπάρχουν). Ακολουθεί το σώμα της κλάσης που περικλείεται σε άγκιστρα. Στο κύριο μέρος μιας κλάσης ορίζονται τα μέλη της. Παρατηρούμε ότι στο παραπάνω παράδειγμα είναι κενό, κάτι που σημαίνει ότι η παραπάνω κλάση είναι μια *κενή κλάση (empty class).* Οι κενές κλάσεις δεν έχουν καμία χρηστικότητα, οπότε πρέπει να προσθέσουμε σε αυτές μέλη. Ας δούμε αναλυτικά ποια είναι τα έγκυρα μέλη μιας κλάσης και πως τα δηλώνουμε.

4.2. Μέλη κλάσης

Τα δεδομένα και οι μέθοδοι μιας κλάσης καλούνται *μέλη κλάσης (class members).* Η επίσημη ορολογία της Microsoft διαχωρίζει τα μέλη σε α) *μέλη δεδομένων (data members)* και β) *μέλη λειτουργιών (function members).* Επιπρόσθετα των δύο αυτών μελών, οι κλάσεις μπορούν να περιέχουν *εμφωλευμένους τύπους* (nested types). Μια κλάση μπορεί να έχει όσα μέλη δεδομένων και μεθόδων χρειάζεται από τους συνδυασμούς 9 πιθανών μελών. Ακολουθεί η περιγραφή των μελών αυτών.

4.2.1. Μέλη δεδομένων

Τα μέλη δεδομένων είναι εκείνα που περιέχουν τα δεδομένα των αντικειμένων μιας κλάσης: τα πεδία (fields), τις σταθερές (constants) και τα γεγονότα (events).

Πεδία

Τα πεδία (fields) είναι η ονομασία που δίνεται στις μεταβλητές που ανήκουν στα αντικείμενα μιας κλάσης ή την ίδια την κλάση (συγκρατήστε αυτόν τον διαχωρισμό γιατί έχει σημασία).

- Μπορεί να είναι οποιουδήποτε τύπου δεδομένων, είτε τύπου τιμής, είτε τύπου αναφοράς.
- Όπως όλες οι μεταβλητές, τα πεδία αποθηκεύουν δεδομένα και έχουν τα ακόλουθα χαρακτηριστικά:
  - Μπορούν να ενημερωθούν.
  - Μπορούν να προσπελαστούν.

Η δήλωση ενός πεδίου με το ελάχιστο συντακτικό είναι η ακόλουθη:

```
Τύπος-δεδομένων προσδιοριστής
```

Ένα παράδειγμα ενός πεδίου μέσα σε μια κλάση, είναι το ακόλουθο, όπου η κλάση MyClass περιέχει μια μεταβλητή int:

```
class MyClass
{
    int MyField;
}
```

Μια πιο πολύπλοκη δήλωση πεδίου θα περιελάμβανε προαιρετικά τα ακόλουθα χαρακτηριστικά :

Πριν τον τύπο δεδομένων του πεδίου	• τον τροποποιητή static • έναν *τροποποιητή πρόσβασης (access modifier).* Αυτοί είναι οι public, protected, internal, private • τον *τροποποιητή κληρονομικότητας* new • τον τροποποιητή readonly • τον *τροποποιητή για τα νήματα* volatile • τον τροποποιητή unsafe
Μετά τον προσδιοριστή	• τον τελεστή ανάθεσης και ένα κυριολεκτικό. • έναν ή περισσότερους προσδιοριστές διαχωρισμένους από κόμματα.

Πρέπει να προσέξουμε ότι αντίθετα με άλλες γλώσσες, στη C# δεν υπάρχουν καθολικές μεταβλητές (global variables), δηλαδή μεταβλητές που να ορίζονται έξω από ένα τύπο. Όλα τα πεδία ανήκουν σε έναν τύπο και πρέπει να δηλώνονται μέσα στον ορισμό του.

Εφόσον ένα πεδίο είναι μια μεταβλητή, μπορεί να αρχικοποιείται και κατά τη δήλωσή του. Ένα πεδίο αρχικοποιείται κατά τη δήλωσή του, αν μετά τον προσδιοριστή του, ακολουθεί ο τελεστής ανάθεσης = και μια έκφραση που αποτιμάται σε μια τιμή του τύπου δεδομένων του πεδίου, ή τουλάχιστον ενός τύπου που μπορεί να μετατραπεί αυτόματα στον τύπο του πεδίου. Η τιμή αυτή πρέπει να μπορεί να προσδιοριστεί κατά τη μεταγλώττιση. Αν δεν ορίζεται τιμή (όπως στο παραπάνω παράδειγμα), η τιμή του πεδίου τίθεται αυτόματα από το μεταγλωττιστή στην προεπιλεγμένη τιμή του τύπου δεδομένων του πεδίου. Οι προεπιλεγμένες τιμές των ενσωματωμένων τύπων της C# δίνονται στους πίνακες 3.8 και 3.11. Για να τους συνοψίσουμε αυτοί είναι οι 0 για τους αριθμητικούς τύπους, false για τον τύπο bool ενώ η προεπιλεγμένη τιμή για τους τύπους αναφοράς είναι η null.

Για παράδειγμα ο κώδικας στο ακόλουθο παράδειγμα περιέχει τέσσερα πεδία. Τα δύο πρώτα πεδία αρχικοποιούνται αυτόματα, ενώ τα δύο τελευταία αρχικοποιούνται κατά τη δήλωσή τους.

```
class MyClass
{
   int f1; // Αρχικοποίηση στην τιμή 0 - value type
   string f2; // Αρχικοποίηση στην τιμή null - reference type
   int f3 = 25; // Αρχικοποίηση στην τιμή 25
   string f4 = "abcd"; // Αρχικοποίηση στην τιμή "abcd"
}
```

Είναι δυνατή επίσης η δήλωση πολλαπλών πεδίων του ίδιου τύπου δεδομένων στην ίδια πρόταση, χωρίζοντας τα ονόματα των πεδίων με κόμματα. Δεν γίνεται να γίνει ταυτόχρονη δήλωση πεδίων διαφορετικών τύπων δεδομένων. Οι δηλώσεις του παραπάνω παραδείγματος μπορούν να συνδυαστούν ως εξής:

```
class MyClass
{
   int f1, f3 = 25;
   string f2, f4 = "abcd";
}
```

Η τιμή ενός πεδίου μπορεί να προσπελαστεί και να ενημερωθεί από τα αντικείμενα της ίδιας της κλάσης ή και άλλων αντικειμένων μέσα από τις μεθόδους τους.

Σταθερές

Μια *σταθερά (constant)* είναι ένα μέλος κλάσης που αναπαριστά μια σταθερή τιμή, δηλαδή μια τιμή που δεν μπορεί να μεταβληθεί κατά τη διάρκεια εκτέλεσης του προγράμματος. Τέτοιου είδους μέλη, μπορούν να φανούν χρήσιμα στην αναγνωσιμότητα του κώδικα, όταν χρειάζεται να εκφραστεί με ένα περιγραφικό όνομα μια τιμή που επαναλαμβάνεται σε διάφορα τμήματα του κώδικα, όπως για παράδειγμα η μαθηματική σταθερά π. Η χρήση των σταθερών, έχει και το προφανές πλεονέκτημα του ότι αν ποτέ χρειαστεί να αλλαχτούν (π.χ. να προστεθεί μεγαλύτερη ακρίβεια στα δεκαδικά ψηφία του π), τότε αλλάζουμε μόνο ένα σημείο στον κώδικά μας. Υπάρχουν δύο ειδών σταθερές: η σταθερές που δηλώνονται με τη λέξη const και οι σταθερές που δηλώνονται με τη λέξη readonly.

Στην περίπτωση χρήσης της const, η τιμή της σταθεράς ορίζεται κατά τη δήλωση της μεταβλητής και πρέπει να μπορεί να αποτιμηθεί κατά τη μεταγλώττιση.

```
class MyMathClass
{
   const double PI = 3.14;
}
```

Στην περίπτωση χρήσης της readonly η τιμή της σταθεράς μπορεί να οριστεί τόσο κατά τη δήλωση της μεταβλητής όσο και μέσα στον κατασκευαστή της κλάσης, δηλαδή κατά τη δημιουργία των αντικειμένων (βλέπε παρακάτω). Αφού όμως οριστεί μια τιμή, αυτή δεν μπορεί να αλλάξει. Τότε λέμε ότι η μεταβλητή γίνεται immutable.

```
class MyMathClass
{
   readonly double PI = 3.14;
   MyMathClass(){
      PI = 3.14159265;
    }
}
```

Γεγονότα

Τα *γεγονότα (events)* είναι μέλη κλάσης που επιτρέπουν σε ένα αντικείμενο να ειδοποιήσει άλλα αντικείμενα ότι συνέβη ένα αξιοπρόσεκτο γεγονός, όπως η αλλαγή μιας τιμής ενός πεδίου ή ότι προκύπτει κάποια διάδραση με το χρήστη (π.χ. το πάτημα ενός κουμπιού). Τα τμήματα κώδικα που χειρίζονται τέτοια γεγονότα, καλούνται χειριστές γεγονότων (event handlers). Τα γεγονότα ορίζονται και ενεργοποιούνται χρησιμοποιώντας τα delegates. Θα αφιερώσουμε ολόκληρη ενότητα στα γεγονότα γιατί συνδέονται με τις παραθυρικές εφαρμογές και είναι αρκετά σημαντικά.

4.2.2. Μέλη λειτουργιών

Τα μέλη λειτουργιών (function members) είναι τα μέλη κλάσεων που παρέχουν τη λειτουργικότητα για την επεξεργασία των δεδομένων της κλάσης (ή και άλλων κλάσεων). Τα μέλη λειτουργιών γενικά μοντελοποιούν τη συμπεριφορά των αντικειμένων του πραγματικού κόσμου.

Μέθοδοι

Οι μέθοδοι (methods) είναι οι γνωστές από άλλες γλώσσες προγραμματισμού συναρτήσεις. Μια μέθοδος είναι ένα ονομασμένο μπλοκ εκτελέσιμου κώδικα που μπορεί να εκτελεστεί από πολλά και διαφορετικά τμήματα ενός προγράμματος, ακόμα και από άλλα προγράμματα.

Όταν μια μέθοδος καλείται (called/invoked), εκτελεί τον κώδικα που περικλείεται στο μπλοκ της και μετά την ολοκλήρωσή της επιστρέφει τη ροή εκτέλεσης εκεί απ' όπου κλήθηκε. Το συντακτικό για τη δήλωση μιας μεθόδου περιλαμβάνει τα ακόλουθα στοιχεία:

- Τροποποιητές: οι τροποποιητές περιλαμβάνουν δεσμευμένες λέξεις που προσδιορίζουν αν είναι δυνατό να κληθεί η μέθοδος από άλλες κλάσεις, εντός και εκτός της ιεραρχίας κλάσεων, εντός και εκτός ενός ονοματοχώρου και εντός και εκτός ενός assembly.

- Τύπο επιστροφής: ο τύπος επιστροφής ορίζει τον τύπο δεδομένων της τιμής που θα επιστρέφει η μέθοδος. Αν μια μέθοδος δεν επιστρέφει κάποια τιμή, ο τύπος επιστροφής ορίζεται ως void.
- Όνομα: ένας προσδιοριστής για να αναγνωρίζεται μοναδικά μια μέθοδος.
- Λίστα παραμέτρων: η λίστα παραμέτρων είναι μια σειρά δηλώσεων παραμέτρων, χωριζόμενων με κόμματα, εσωκλειόμενων σε παρενθέσεις. Κάθε δήλωση παραμέτρου είναι ένα ζευγάρι τύπου δεδομένων – προσδιοριστή, παρόμοια δηλαδή με τη δήλωση μεταβλητών.
- Σώμα μεθόδου: το σώμα αποτελείται από εκτελέσιμο κώδικα κλεισμένο μέσα σε άγκιστρα.

```
<τροποποιητές> τύπος_επιστροφής όνομα-μεθόδου(<λίστα-παραμέτρων>)
{
// Σώμα μεθόδου
}
```

Για παράδειγμα, ο επόμενος κώδικας ορίζει μια κλάση με μια μέθοδο με το όνομα PrintNums.

```
class SimpleClass
{
   void PrintNums()
   {
      Console.WriteLine("1");
      Console.WriteLine("2");
   }
}
```

Από τη δήλωση, μπορούμε να πούμε τα ακόλουθα για την PrintNums:

- Δεν επιστρέφει κάποια τιμή, άρα ο τύπος επιστροφής ορίζεται σαν void.
- Έχει μια κενή λίστα παραμέτρων.
- Περιέχει δυο εντολές μέσα στο κύριο μέρος της μεθόδου.

Αντίθετα με άλλες γλώσσες προγραμματισμού, δεν γίνεται να οριστούν καθολικές μέθοδοι (global functions), δηλαδή μέθοδοι που ορίζονται έξω από μια δήλωση κλάσης (ή τύπου γενικότερα).

Στο σώμα μιας μεθόδου είναι ένα μπλοκ κώδικα, το οποίο όπως αναφέραμε είναι μια σειρά εντολών ανάμεσα σε άγκιστρα. Το μπλοκ αυτό μπορεί να περιέχει τα ακόλουθα συστατικά:

- Τοπικές μεταβλητές
- Προτάσεις ελέγχου ροής
- Κλήσεις άλλων μεθόδων (ή και της ίδιας οπότε έχουμε αναδρομή)
- Άλλα μπλοκ κώδικα

Μια μέθοδος μπορεί να καλέσει άλλες μεθόδους από το σώμα της. Η κλήση μιας μεθόδου γίνεται απλώς με το όνομά της και τη λίστα παραμέτρων της. Κατά την κλήση μιας μεθόδου:

1. Η εκτέλεση της τρέχουσας μεθόδου παύει στο σημείο της κλήσης.
2. Η ροή εκτέλεσης μεταφέρεται στην αρχή της μεθόδου που κλήθηκε.
3. Η καλεσμένη μέθοδος εκτελείται μέχρι να ολοκληρώσει.
4. Η ροή επιστρέφει στην αρχική μέθοδο.

Μια μέθοδος μπορεί να επιστρέψει μια τιμή στον κώδικα που την καλεί στη θέση της έκφρασης απ’ όπου κλήθηκε. Για να επιστραφεί μια τιμή η μέθοδος πρέπει να δηλώσει έναν τύπο επιστροφής πριν το όνομά της και να επιστρέψει μια τιμή με την εντολή return. Αν η μέθοδος δεν επιστρέφει τιμή πρέπει να δηλωθεί void.

Μια κλάση μπορεί να έχει περισσότερες από μια μεθόδους με το ίδιο όνομα. Το χαρακτηριστικό αυτό καλείται *υπερφόρτωση μεθόδου (overloading).* Κάθε μέθοδος με το ίδιο όνομα πρέπει να έχει διαφορετική υπογραφή από τις υπόλοιπες μεθόδους.

Η υπογραφή (signature) μιας μεθόδου αποτελείται από τις ακόλουθες πληροφορίες:

- Το όνομα μιας μεθόδου
- Το πλήθος των παραμέτρων της

- Τον τύπο δεδομένων των παραμέτρων και τη σειρά εμφάνισης τους
- Τους τροποποιητές των παραμέτρων

Ο τύπος επιστροφής μιας μεθόδου δεν είναι μέρος της υπογραφής της, όπως επίσης και τα ονόματα των παραμέτρων της. Ακολουθεί ένα παράδειγμα υπερφόρτωσης μεθόδου:

```
class Overload {
   public void OverloadedMethod() {
      Console.WriteLine("No parameters");
   }
   // Overload με μια ακέραια παράμετρο.
   public void OverloadedMethod(int a) {
      Console.WriteLine("One parameter: " + a);
   }
   // Overload με μια παράμετρο συμβολοσειρών.
   public void OverloadedMethod(string a) {
      Console.WriteLine("One parameter: " + a);
   }

   // Overload για δύο ακέραιες παραμέτρους.
   public int OverloadedMethod(int a, int b) {
      Console.WriteLine("Two parameters: " + a + " " + b);
      return a + b;
   }
   // Overload για δυο double παραμετρους.
   public double OverloadedMethod(double a, double b) {
      Console.WriteLine("Two double parameters: " + a + " "+ b);
      return a + b;
   }
}

class OverloadDemo {
   static void Main() {
      Overload ob = new Overload();
      int resI;
      double resD;
      //Κάλεσμα όλων των μεθόδων της OverloadedMethod().
      ob.OverloadedMethod();
      Console.WriteLine();
      ob.OverloadedMethod(2);
      Console.WriteLine();
      ob.OverloadedMethod("Hello!");
      Console.WriteLine();
      resI = ob.OverloadedMethod (4, 6);
      Console.WriteLine("Result of ob.OverloadedMethod(4, 6): " + resI);
      Console.WriteLine();
      resD = ob.OverloadedMethod(1.1, 2.32);
      Console.WriteLine("Result of ob.OverloadedMethod(1.1, 2.32): " + resD);
  }
}
```

Οι παράμετροι μιας μεθόδου στη C# είναι ειδικές μεταβλητές που επιτρέπουν τόσο τη πέρασμα δεδομένων στη μέθοδο κατά την έναρξη της εκτέλεσής της, όσο και την έξοδο δεδομένων από αυτή κατά τον τερματισμό της. Οι *τυπικές παράμετροι(formal parameters)* είναι τοπικές μεταβλητές που δηλώνονται στη λίστα παραμέτρων της μεθόδου, αντί να δηλώνονται στο σώμα της. Αντίθετα όμως με τις τοπικές μεταβλητές, οι τυπικές παράμετροι ορίζονται έξω από το σώμα της μεθόδου και αρχικοποιούνται πριν το ξεκίνημα της εκτέλεσής της, εκτός από τις *παραμέτρους εξόδου (output parameters)*. Η λίστα παραμέτρων μπορεί να έχει οποιοδήποτε αριθμό τυπικών παραμέτρων και οι δηλώσεις πρέπει να διαχωρίζονται με κόμματα. Οι τυπικές παράμετροι χρησιμοποιούνται στο σώμα της μεθόδου, όπως κάθε άλλη τοπική μεταβλητή.

Όταν καλείται μια μέθοδος μέσα από τον κώδικα, οι τιμές των τυπικών παραμέτρων πρέπει να αρχικοποιηθούν πριν το ξεκίνημα της εκτέλεσης των παραμέτρων. Οι εκφράσεις που χρησιμοποιούνται για την αρχικοποίηση των τυπικών παραμέτρων καλούνται *πραγματικές παράμετροι (actual parameters).* Μερικές φορές καλούνται και *ορίσματα (arguments).* Οι πραγματικές παράμετροι τοποθετούνται στην λίστα παραμέτρων της μεθόδου κατά την κλήση της. Κάθε πραγματική παράμετρος πρέπει να έχει τον τύπο της αντίστοιχης τυπικής παραμέτρου, ή ο μεταγλωττιστής θα πρέπει να μπορεί να μετατρέψει τον τύπο της στον τύπο της τυπικής παραμέτρου.

Οι τύποι παραμέτρων που είδαμε ως τώρα καλούνται *παράμετροι τιμής (value parametrs)*, γιατί τα δεδομένα που περνούν στη μέθοδο από αυτές *αντιγράφουν* την τιμή της πραγματικής παραμέτρου στην τυπική παράμετρο. Όταν καλείται μια μέθοδος, το σύστημα δεσμεύει χώρο στη μνήμη (και πιο συγκεκριμένα στο σωρό) για τις τυπικές παραμέτρους και αντιγράφει τις τιμές των πραγματικών παραμέτρων σε αυτές. Επομένως κάθε αλλαγή στις τιμές των τυπικών παραμέτρων *δεν επηρεάζει τις τιμές των πραγματικών παραμέτρων*. Αυτή η ιδιότητα είναι γνωστή στη βιβλιογραφία και ως *πέρασμα με τιμή (pass by value).*

Το δεύτερο είδος παραμέτρων που υπάρχει είναι οι *παράμετροι αναφορών (reference parameters)*. Όταν χρησιμοποιούμε μια παράμετρο αναφοράς πρέπει να χρησιμοποιήσουμε τον τροποποιητή *ref*, τόσο κατά τη δήλωση της μεθόδου όσο και κατά την κλήση της. Η πραγματική τιμή *πρέπει* να είναι μεταβλητή (άρα όχι κυριολεκτικό) και πρέπει να είναι αρχικοποιημένη πριν χρησιμοποιηθεί σαν πραγματική παράμετρος. Αν είναι μεταβλητή τύπου αναφοράς τότε είτε πρέπει να αναφέρεται σε ένα πραγματικό αντικείμενο, είτε να έχει την τιμή null. Σε αντίθεση με τις παραμέτρους τιμών, στις παραμέτρους αναφορών η τυπική παράμετρος συμπεριφέρεται σαν *ψευδώνυμο (alias)* της πραγματικής παραμέτρου, δηλαδή αναφέρεται στην ίδια θέση μνήμης με την πραγματική παράμετρο. Κατά συνέπεια, κάθε αλλαγή στις τιμές των τυπικών παραμέτρων *επηρεάζει τις τιμές των πραγματικών παραμέτρων.* Αυτή η ιδιότητα είναι γνωστή στη βιβλιογραφία και ως *πέρασμα με αναφορά (pass by reference)*.

Το τρίτο είδος παραμέτρων που μπορεί να υπάρξει σε μια μέθοδο είναι οι *παράμετροι εξόδου (output parameters)* και χρίζει ιδιαίτερης προσοχής. Η συμπεριφορά τους είναι ίδια με την συμπεριφορά των παραμέτρων αναφορών, ωστόσο δεν χρειάζεται να είναι αρχικοποιημένη η πραγματική παράμετρος όπως στις παραμέτρους αναφορών, ενώ *δεν πρέπει* να είναι οποιαδήποτε άλλη έκφραση, έστω και αν αποτιμάται σε κάποια τιμή. Επίσης μέσα στο σώμα της μεθόδου *πρέπει υποχρεωτικά να αρχικοποιηθεί η τιμή της παραμέτρου πριν αυτή προσπελαστεί*. Κάθε μετέπειτα αλλαγή στην τυπική παράμετρο αντικατοπτρίζεται και στην πραγματική παράμετρο. Δηλώνουμε μια παράμετρο εξόδου με τον τροποποιητή out. Επίσης καλούμε μια μέθοδο με παράμετρο εξόδου πάλι με τον τροποποιητή out. Ακολουθεί ένα παράδειγμα:

```
class Program
{
   static void MyMethod(out MyClass f1, out int f2)
   {
      f1 = new MyClass(); // δημιουργία αντικειμένου της κλάσης.
      f1.Val = 25; // Ανάθεση στο πεδίο της κλάσης.
      f2 = 15; // ανάθεση στη δεύτερη παράμετρο int.
   }
   static void Main()
   {
      MyClass a1 = null;
      int a2;
      MyMethod(out a1, out a2); // κλήση της μεθόδου
   }
}
```

Για τα παραπάνω είδη παραμέτρων που είδαμε πρέπει να υπάρχει ακριβώς μια πραγματική παράμετρος για κάθε τυπική. Οι *πίνακες παραμέτρων (parameter arrays)* είναι διαφορετικοί στο ότι επιτρέπουν την ύπαρξη πολλαπλών πραγματικών παραμέτρων για μια συγκεκριμένη τυπική παράμετρο. Για τους πίνακες παραμέτρων πρέπει να προσέξουμε τα εξής σημεία:

- Μπορεί να υπάρξει μόνο ένας πίνακας παραμέτρων σε μια λίστα παραμέτρων.
- Αν υπάρχει πίνακας παραμέτρων σε μια λίστα παραμέτρων, πρέπει να είναι η τελευταία παράμετρος στη λίστα.

Ένας πίνακας παραμέτρων δηλώνεται χρησιμοποιώντας τον τροποποιητή params πριν τον τύπο δεδομένων της παραμέτρου, ενώ χρειάζονται άδειες αγκύλες μετά τον τύπο δεδομένων. Μια τέτοια δήλωση θα έμοιαζε σαν την ακόλουθη:

```
void ListInts( params int[] inVals ) {}
```

Οι αγκύλες δίπλα από τον τύπο δεδομένων προσδιορίζουν ότι η παράμετρος θα είναι ένας πίνακας ακεραίων (array of ints). Οι πίνακες θα αναλυθούν σε επόμενη ενότητα αλλά το μόνο που χρειαζόμαστε να ξέρουμε για αυτούς προς το παρόν είναι ότι είναι μια διατεταγμένη λίστα μεταβλητών του ίδιου τύπου δεδομένων που προσπελαύνεται με έναν αριθμητικό δείκτη.

Τα τέσσερα είδη παραμέτρων που είδαμε συνοψίζονται στον πίνακα 4.1.

Πίνακας 4.1 - Είδη παραμέτρων σε μια μέθοδο

Είδος παραμέτρου	Τροποποιητής	Χρησιμοποιείται κατά τη δήλωση;	Χρησιμοποιείται κατά την κλήση;	Υλοποίηση
Value	Κανένας	-	-	Το σύστημα αντιγράφει την τιμή της πραγματικής παραμέτρου στην τυπική παράμετρο
Reference	ref	Ναι	Ναι	Η τυπική παράμετρος αποτελεί ψευδώνυμο της πραγματικής παραμέτρου
Output	out	Ναι	Ναι	Η τυπική παράμετρος αποτελεί ψευδώνυμο της πραγματικής παραμέτρου
Array	params	Ναι	Όχι	Περνάει ένα πλήθος πραγματικών παραμέτρων στη μέθοδο

Τέλος η C# μας δίνει δυο χαρακτηριστικά για τις παραμέτρους που μπορούν να μας φανούν χρήσιμα. Αυτά είναι οι *προαιρετικές παράμετροι (optional parameters)* και οι *ονομασμένες παράμετροι (named parameters).* Οι πρώτες μας επιτρέπουν να παραλείψουμε την παροχή μιας παραμέτρου κατά την κλήση μιας μεθόδου και οι δεύτερες μας επιτρέπουν να αλλάξουμε τη σειρά των παραμέτρων κατά την κλήση μιας μεθόδου. Το πρώτο επιτυγχάνεται κάνοντας μια ανάθεση τιμής σε μια τυπική παράμετρο μιας μεθόδου και το δεύτερο επιτυγχάνεται παρέχοντας το όνομα της τυπικής παραμέτρου ακολουθούμενο από άνω και κάτω τελεία κατά την κλήση της μεθόδου. Ακολουθεί ένα παράδειγμα:

```
class  Calculator
{
   public int PerformCalculation(int x, int y = 10, int divisor = 2)
   {
      return (x*y)/divisor;
   }
   static void Main(string[] args)
   {
      Calculator c = new Calculator(); //default constructor
      //παρακάτω περνάμε μόνο δυο παραμέτρους και μάλιστα σε διαφορετική
      //σειρά απ' ότι δηλώθηκαν
      //παρατηρήστε ότι το x είναι υποχρεωτικό και δεν γίνεται να παραληφθεί
      int result = c.PerformCalculation(y: 10, x: 1);
   }
}
```

Ιδιότητες

Οι ιδιότητες (properties) μοιάζουν με πεδία εξωτερικά, αλλά εσωτερικά περιέχουν λογική, δηλαδή εκτελέσιμο κώδικα, ακριβώς όπως οι μέθοδοι. Οι ιδιότητες μπορούν να χρησιμοποιηθούν με τον ίδιο τρόπο που χρησιμοποιούνται τα μέλη δεδομένων. Η C# παρέχει ένα συγκεκριμένο συντακτικό για την υλοποίηση ιδιοτήτων για το διάβασμα (read) και το γράψιμο (write) των πεδίων μιας κλάσης, για να μη χρειάζεται η χρήση μεθόδων με το όνομα getX ή setY, όπως χρειάζεται σε άλλες γλώσσες προγραμματισμού.

Μια ιδιότητα δηλώνεται όπως ένα πεδίο, αλλά με την προσθήκη δυο get και set μπλοκ κώδικα. Ακολουθεί ένα παράδειγμα:

```
public class MyClass
{
   decimal price;
   public decimal Price
   {
```

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

Οι λέξεις get και set καλούνται property accessors. Ο get accessor εκτελείται όταν η ιδιότητα διαβάζεται. Πρέπει να επιστρέφει μια τιμή του τύπου δεδομένων της ιδιότητας. Ο set accessor εκτελείται όταν ανατίθεται τιμή. Έχει μια νοητή παράμετρο με το όνομα value του ίδιου τύπου δεδομένων με αυτού της ιδιότητας, η οποία ανατίθεται στο πεδίο που αντιστοιχεί στην ιδιότητα. Στο παραπάνω παράδειγμα η νοητή παράμετρος value αναθέτει μια τιμή στο πεδίο price.

Παρόλο που οι ιδιότητες προσπελαύνονται με τον ίδιο τρόπο που προσπελαύνονται τα πεδία, διαφέρουν στο γεγονός ότι δίνουν πλήρη έλεγχο στον προγραμματιστή στο πως θα παρουσιάσει τα δεδομένα χωρίς να εκθέσει τις εσωτερικές τους λεπτομέρειες. Στο παράδειγμα παραπάνω, θα μπορούσε να προκύπτει μια εξαίρεση, ή να εμφανίζεται ένα μήνυμα λάθους αν η value ήταν π.χ. αρνητική τιμή (αφού ένα κόστος δεν μπορεί να έχει αρνητική τιμή).

Οι ιδιότητες μπορούν να έχουν μπροστά από τον τύπο δεδομένων τους, τους εξής τροποποιητές:

- Τον τροποποιητή static
- Τους τροποποιητές πρόσβασης: public, internal, private, protected
- Τους τροποποιητές κληρονομικότητας: new, virtual, abstract, override, sealed
- Τους τροποποιητές μη-διαχειρισμένου κώδικα: unsafe, extern

Μια ιδιότητα καλείται read-only αν ορίζει μόνο έναν get accessor και είναι write-only αν ορίζει μόνο έναν set accessor. Οι ιδιότητες write-only χρησιμοποιούνται σπάνια. Μια ιδιότητα συνήθως αντιστοιχεί αποκλειστικά σε ένα πεδίο που αποθηκεύει τα δεδομένα της. Ωστόσο είναι δυνατό, μια ιδιότητα να υπολογίζεται από άλλα δεδομένα, όπως στο παρακάτω παράδειγμα:

```
decimal currentPrice, sharesOwned;
public decimal Worth
{
    get { return currentPrice * sharesOwned; }
}
```

Η πιο συνηθισμένη υλοποίηση μιας ιδιότητας που αποτελεί ταυτόχρονα και καλή πρακτική είναι ένα get και ένα set που απλώς διαβάζουν και γράφουν σε ένα (private) πεδίο του ίδιου τύπου δεδομένων με την ιδιότητα. Για τον λόγο αυτό, υπάρχει η δυνατότητα χρήσης της *αυτόματης ιδιότητας (automatic property).* Μια δήλωση αυτόματης ιδιότητας ζητάει από το μεταγλωττιστή να ορίσει αυτόματα ένα πεδίο που αντιστοιχεί στην ιδιότητα. Ένα παράδειγμα θα ήταν το ακόλουθο:

```
public class Stock
{
    //... άλλες δηλώσεις
    public decimal CurrentPrice { get; set; }
}
```

Ο μεταγλωττιστής παράγει αυτόματα ένα αντίστοιχο πεδίο με ένα όνομα που δεν μπορεί ωστόσο να χρησιμοποιηθεί. Η χρήση του προφανώς θα γίνεται μόνο μέσω της ιδιότητας. Ακόμα και στις αυτόματες ιδιότητες είναι δυνατόν να οριστούν τροποποιητές πριν τις λέξεις get και set για τον έλεγχο πρόσβασης στο διάβασμα ή το γράψιμο.

Κατασκευαστές

Οι κατασκευαστές (constructors) είναι ειδικές μέθοδοι που καλούνται κατά τη δημιουργία ενός αντικειμένου. Πρέπει να έχουν το ίδιο όνομα με την κλάση στην οποία ανήκουν και δεν μπορεί να έχουν τύπο επιστροφής. Οι κατασκευαστές παρουσιάζονται αναλυτικά στην παράγραφο 4.3.

Καταστροφείς

Οι καταστροφείς είναι παρόμοιοι με τους κατασκευαστές αλλά καλούνται όταν το CLR αντιληφθεί ότι ένα αντικείμενο δεν χρησιμοποιείται και κατά συνέπεια δεν χρειάζεται άλλο στο πρόγραμμα. Οι καταστροφείς έχουν το ίδιο όνομα με την κλάση στην οποία ανήκουν με μια πεσπιρωμένη (tilde) μπροστά (~).

Τελεστές

Η C# επιτρέπει την παράκαμψη της λειτουργικότητας των τελεστών μέσα σε μια κλάση, από την προεπιλεγμένη συμπεριφορά τους όπως τους είδαμε.

Indexers

Οι indexers επιτρέπουν τη δημιουργία ευρετηρίων από τα μέλη μιας κλάσης, δίνοντάς τους λειτουργικότητα όμοια με αυτή των πινάκων ή των συλλογών. Έστω για παράδειγμα ότι ορίζουμε μια κλάση με τρία πεδία. Μπορούμε να χρησιμοποιήσουμε αυτά τα πεδία έξω από την κλάση με τη σημειογραφία dot-syntax (περιγράφεται στην επόμενη παράγραφο 4.2.3). Αυτό φαίνεται στον ακόλουθο κώδικα:

```
class MyClass
{
   public int a;
   public int b;
   public int c;
}
class Program
{
   static void Main()
   {
      MyClass obj = new MyClass();
      obj.a = 1;
      obj.b = 2;
      obj.c = 3;
   }
}
```

Υπάρχουν περιπτώσεις ωστόσο που είναι βολικό να προσπελαύνουμε τα πεδία αυτά με τον ίδιο τρόπο που προσπελαύνουμε τους πίνακες, δηλαδή με τη *σημειογραφία ευρετηρίου(index notation).* Η σημειογραφία αυτή αποτελείται από έναν δείκτη ανάμεσα σε αγκύλες. Ακολουθεί ένα παράδειγμα που χρησιμοποιεί τη MyClass όπως ορίστηκε παραπάνω:

```
class Program
{
   static void Main()
   {
      MyClass obj = new MyClass();
      obj[1] = 1; //ίδια δήλωση με obj.a = 1;
      obj[2] = 2; //ίδια δήλωση με obj.a = 2;
      obj[3] = 3; //ίδια δήλωση με obj.a = 3;
   }
}
```

Ένας indexer είναι ένα ζευγάρι από set και get accessors, παρόμοιους με αυτούς των ιδιοτήτων. Οι indexers και οι ιδιότητες είναι παρόμοιες σε πολλαπλά σημεία:

- Ένας indexer δεν δεσμεύει μνήμη για αποθήκευση.
- Τόσο οι indexers όσο και οι ιδιότητες χρησιμοποιούνται κυρίως για να δίνουν πρόσβαση σε άλλα μέλη δεδομένων μιας κλάσης με τα οποία συσχετίζονται και για τα οποία παρέχουν τα get και set.

- Μια ιδιότητα κυρίως αναπαριστά ένα μέλος δεδομένων.
- Ένας indexer συνήθως αναπαριστά πολλαπλά μέλη δεδομένων.

Πρέπει να παρατηρήσουμε τα εξής για το συντακτικό δήλωσης ενός indexer:

- Ένας indexer δεν έχει όνομα αλλά χρησιμοποιεί τη λέξη this.
- Η λίστα παραμέτρων βρίσκεται ανάμεσα σε αγκύλες.
- Πρέπει να υπάρχει τουλάχιστον μια παράμετρος στη λίστα παραμέτρων.

Το συντακτικό δήλωσης θα φαίνεται κάπως έτσι:

```
ReturnType this [ Type param1, ... ]
{
   get Square bracket Square bracket
   {
      //...
   }
   set
   {
      //...
   }
}
```

4.2.3. Προσπέλαση μελών κλάσης

Μέσα σε μια κλάση, μέσα σε κάθε μέλος λειτουργίας όπως για παράδειγμα μια μέθοδο, μπορεί να προσπελαστεί οποιοδήποτε άλλο μέλος της κλάσης, πολύ απλά χρησιμοποιώντας το όνομά του. Ήδη έχουμε δει μερικά παραδείγματα παραπάνω αλλά ας ξαναδούμε ένα παράδειγμα:

```
public class MyClass
{
   decimal myField;
   public decimal Price
   {
      get
      {
         return myField * 2;  //προσπέλαση του πεδίου myField
      }
      set
      {
         myField = value / 2; //προσπέλαση του πεδίου myField
      }
   }
   void PrintNum()
   {
      Console.WriteLine(myField); //προσπέλαση του πεδίου myField
   }

}
```

Αυτό που χρειάζεται προσοχή ωστόσο είναι η προσπέλαση των μελών εκτός της κλάσης που ορίζονται. Στην περίπτωση αυτή μιλάμε πλέον για προσπέλαση μελών αντικειμένων και πιο συγκεκριμένα για προσπέλαση *μελών στιγμιότυπου (instance members)*. Η προσπέλαση των μελών στιγμιότυπου γίνεται μόνο μέσα από τα αντικείμενα μιας κλάσης, κάτι που σημαίνει ότι πρώτα πρέπει να έχουμε δημιουργήσει ένα αντικείμενό της (παράγραφος 4.3). Αφού έχουμε διαθέσιμο ένα αρχικοποιημένο αντικείμενο μπορούμε να προσπελάσουμε τα διαθέσιμα μέλη της με τον τελεστή τελεία (.). Η χρήση της τελείας για την προσπέλαση των μελών στιγμιότυπου εκτός της κλάσης όπου ορίζονται καλείται *σημειογραφία dot-syntax*. Η σημειογραφία αυτή αποτελείται από το όνομα του αντικειμένου (στιγμιότυπου), ακολουθούμενο από μια τελεία, ακολουθούμενο από το όνομα του μέλους. Μια πρόταση που προσπελαύνει ένα μέλος στιγμιότυπου είναι η ακόλουθη:

```
myObject.myField = 5;
```

Η παραπάνω πρόταση για να είναι έγκυρη, προϋποθέτει την ύπαρξη ενός αρχικοποιημένου αντικειμένου, στην συγκεκριμένη περίπτωση του myObject και την ύπαρξη ενός *προσβάσιμου (accessible)* πεδίου. Όπως θα δούμε σε επόμενη ενότητα, μπορούμε να ελέγξουμε το αν θα είναι προσβάσιμο ένα μέλος από μια κλάση εξωτερικά, από μια κλάση ενός άλλου ονοματοχώρου ή από μια κλάση εκτός του assembly. Η C# μας παρέχει έναν ολοκληρωμένο τρόπο για τον έλεγχο πρόσβασης (access control) στα μέλη μιας κλάσης με τους *τροποποιητές πρόσβασης (access modifiers)*.

Η λέξη this

Η λέξη this, που χρησιμοποιείται μέσα σε μια κλάση, είναι μια αναφορά στο τρέχον αντικείμενο. Μπορεί να χρησιμοποιηθεί μόνο μέσα σε μπλοκ κώδικα των ακόλουθων μελών κλάσης:

- Των κατασκευαστών στιγμιότυπου
- Των μεθόδων στιγμιότυπου
- Των ιδιοτήτων και των indexers

Η λέξη this χρησιμοποιείται κυρίως για να διακρίνουμε τα μέλη μιας κλάσης από συνώνυμες τοπικές μεταβλητές ή παραμέτρους ή απλώς σαν μια παράμετρος σε μια μέθοδο. Για παράδειγμα στον παρακάτω κώδικα, δηλώνεται μια κλάση με ένα πεδίο int και μια μέθοδο που λαμβάνει μια μόνο παράμετρο. Η μέθοδος συγκρίνει τις τιμές της παραμέτρου και του πεδίου και επιστρέφει αυτήν που έχει τη μεγαλύτερη τιμή. Το μόνο περίπλοκο σε αυτό το παράδειγμα είναι ότι το όνομα του πεδίου και της μεταβλητής είναι το ίδιο. Για να διακρίνουμε τις δύο μεταβλητές μεταξύ τους χρησιμοποιούμε τη λέξη this.

```
class MyClass
{
   int Var1 = 10;
   public int ReturnMaxSum(int Var1)
   {
      return Var1 > this.Var1 ? Var1 : this.Var1;
   }
}
class Program
{
   static void Main()
   {
      MyClass mc = new MyClass();
      Console.WriteLine("Max: {0}", mc.ReturnMaxSum(30));
      Console.WriteLine("Max: {0}", mc.ReturnMaxSum(5));
   }
}
```

4.3. Δημιουργία αντικειμένων

Έχοντας μια λειτουργική δήλωση κλάσης το επόμενο βήμα είναι να δημιουργήσουμε αντικείμενα από αυτή και να τα χρησιμοποιήσουμε στον κώδικά μας. Η δημιουργία ενός αντικειμένου από μια κλάση αποτελείται από δυο χαρακτηριστικά:

- Τον τελεστή new
- Έναν κατασκευαστή της κλάσης

Ας εξετάσουμε καθένα από τα χαρακτηριστικά αυτά ξεχωριστά.

4.3.1. Κατασκευαστές

Οι κατασκευαστές (constructors) είναι ειδικές μέθοδοι που καλούνται κατά τη δημιουργία ενός αντικειμένου. Οι κατασκευαστές περιέχουν τις εντολές αρχικοποίησης ενός αντικειμένου, δηλαδή τις εντολές που δίνουν στο αντικείμενο μια αρχική κατάσταση (initial state). Τέτοιες μπορεί να είναι η ανάθεση αρχικών τιμών στα πεδία του αντικειμένου, το άνοιγμα συνδέσεων στο δίκτυο ή σε βάσεις δεδομένων και άλλες.

Το συντακτικό για τη δήλωση ενός απλού κατασκευαστή είναι μια μέθοδος που έχει το ίδιο όνομα με την κλάση που τον περιέχει αλλά δεν έχει τύπο επιστροφής:

```
public class MyClass
{
```

```
   public MyClass()
   {
   }
   //... υπόλοιπη δήλωση της κλάσης
}
```

Οι κατασκευαστές επιτρέπουν τους ακόλουθους τροποποιητές πριν το όνομά τους:

- Τους τροποποιητές πρόσβασης public, internal, private, protected
- Τους τροποποιητές μη-διαχειρισμένου κώδικα: unsafe, extern

Σε μια κλάση μπορούν να οριστούν περισσότεροι από ένας κατασκευαστές. Οι κατασκευαστές διακρίνονται μεταξύ τους ανάλογα με το πλήθος και τον τύπο δεδομένων των παραμέτρων τους. Για κάθε πιθανό συνδυασμό παραμέτρων όμως μπορεί να οριστεί μόνο ένας κατασκευαστής. Η δυνατότητα αυτή ονομάζεται *υπερφόρτωση κατασκευαστών (constructor overloading).* Ακολουθεί ένα παράδειγμα:

```
class MyClass
{
   int Id;
   string Name;
   public MyClass() { Id=28; Name="Nemo"; } // Constructor 0
   public MyClass(int val) { Id=val; Name="Nemo"; } // Constructor 1
   public MyClass(string name) { Name=name; } // Constructor 2
   public void PrintName()
   {
       Console.WriteLine("Name {0}, Id {1}", Name, Id);
   }
}
```

Στο παραπάνω παράδειγμα η κλάση MyClass περιέχει τρεις κατασκευαστές: έναν χωρίς παραμέτρους, έναν με μια παράμετρο και πιο συγκεκριμένα έναν ακέραιο και έναν πάλι με μια παράμετρο αλλά αυτή τη φορά τύπου string. Παρατηρούμε ότι οι παράμετροι των κατασκευαστών έχουν το ίδιο όνομα με το όνομα των πεδίων, αλλά αν προσέξουμε πιο προσεκτικά τα ονόματα των πεδίων ξεκινούν με κεφαλαίο. Επειδή η C# κάνει διάκριση μεταξύ πεζών και κεφαλαίων, είναι σε θέση να διακρίνει τα πεδία από τις παραμέτρους. Στην περίπτωση ωστόσο που είχαν ακριβώς το ίδιο όνομα τότε θα υπήρχε πρόβλημα και ο μεταγλωττιστής θα αναγνώριζε μόνο την τοπική μεταβλητή.

Ένας κατασκευαστής μπορεί να αρχικοποιεί ένα σύνολο πεδίων ακριβώς με τον ίδιο τρόπο που αρχικοποιεί το ίδιο σύνολο πεδίων ένας άλλος κατασκευαστής, αλλά να διαφοροποιείται στην αρχικοποίηση ορισμένων μόνο μεταβλητών. Η C# δίνει τη δυνατότητα κλήσης ενός κατασκευαστή από έναν άλλο με τη χρήση της λέξης this, για την αποφυγή της επανάληψης κώδικα. Η δυνατότητα αυτή φαίνεται στο ακόλουθο παράδειγμα:

```
public class Wine
{
   public decimal Price;
   public int Year;
   public Wine (decimal price) { Price = price; }
   public Wine (decimal price, int year) : this (price) { Year = year; }
}
```

Αν δεν έχει οριστεί κάποιος κατασκευαστής σε μια κλάση, ο μεταγλωττιστής παράγει αυτόματα έναν κατασκευαστή χωρίς παραμέτρους. Αν όμως δηλωθεί έστω και ένας με οποιεσδήποτε παραμέτρους τότε αυτός ο κατασκευαστής δεν δημιουργείται.

Άλλο ένα λιγότερο δημοφιλές χαρακτηριστικό των κατασκευαστών, είναι η αρχικοποίηση των παραμέτρων μέσα στη λίστα των παραμέτρων του κατασκευαστή, καθιστώντας τες προαιρετικές. Για παράδειγμα:

```
public Student(string name, bool hasGraduated = false, bool isFreshman = false)
{
   Name = name;
   HasGraduated = hasGraduated;
   IsFreshman = isFreshman;
}
```

Με αυτή τη δήλωση μπορεί να παραληφθεί οποιαδήποτε εκ των αρχικοποιημένων παραμέτρων κατά την κλήση του κατασκευαστή. Ωστόσο μια τέτοια δήλωση καλό είναι να αποφεύγεται γιατί δημιουργεί προβλήματα μεταξύ διαφορετικών assemblies κατά τις αλλαγές της λίστας παραμέτρων.

4.3.2. Δήλωση αντικειμένων

Η δήλωση μιας κλάσης απλώς περιγράφει τη δομή των αντικειμένων της. Έχοντας δηλώσει μια κλάση και (προαιρετικά) έναν ή περισσότερους κατασκευαστές, μπορούμε να δημιουργήσουμε τα αντικείμενα μιας κλάσης. Η δήλωση ενός αντικειμένου, δηλαδή μιας μεταβλητής με τύπο δεδομένων μια κλάση (τύπο αναφοράς δηλαδή), γίνεται με τον ίδιο τρόπο που δηλώνονται οι μεταβλητές τύπου τιμής:

```
class MyClass{} //δήλωση κλάσης

class Program
{
   static void Main()
   {
      MyClass myObject; //δήλωση αντικειμένου, δηλ. μεταβλητής τύπου αναφοράς
   }
}
```

Οι κλάσεις είναι *τύποι αναφορών (reference types)*, κάτι που σημαίνει ότι χρειάζονται μνήμη τόσο για την αναφορά προς τα δεδομένα, όσο και για τα δεδομένα καθαυτά. Η αναφορά προς τα δεδομένα είναι ουσιαστικά μια διεύθυνση μνήμης που αποθηκεύεται σε μια μεταβλητή του τύπου της κλάσης. Για να δημιουργηθεί λοιπόν ένα αντικείμενο μιας κλάσης πρέπει να ξεκινήσουμε δηλώνοντας μια αναφορά. Αν αυτή η αναφορά (δηλαδή η μεταβλητή του τύπου της κλάσης) δεν έχει αρχικοποιηθεί, η τιμή της είναι *μη καθορισμένη(undefined)*. Στο παραπάνω παράδειγμα λοιπόν η μεταβλητή myObject, που έτσι όπως είναι, είναι κενή, θα κρατήσει μια αναφορά και όχι τα μέλη δεδομένων του αντικειμένου, όπως προστάζει η κλάση. Αυτή η διάκριση είναι πολύ σημαντική και πρέπει να τη θυμόμαστε πάντα. Για να δεσμεύσουμε τη μνήμη που θα αποθηκεύει τα πραγματικά δεδομένα του αντικειμένου πρέπει να χρησιμοποιήσουμε τον τελεστή new.

4.3.3. Ο τελεστής new

Η δήλωση μιας μεταβλητής τύπου αναφοράς, δεσμεύει μεν χώρο στη μνήμη για μια αναφορά, αλλά όχι και για τα πραγματικά δεδομένα του αντικειμένου. Για τη δέσμευση των πραγματικών δεδομένων χρησιμοποιούμε τον τελεστή new. Ο τελεστής new δεσμεύει μνήμη και αρχικοποιεί ένα στιγμιότυπο ενός συγκεκριμένου τύπου αναφοράς, καλώντας τον κατασκευαστή του. Μια έκφραση *δημιουργίας αντικειμένων (object creation expression)* λοιπόν έχει την εξής μορφή:

```
new όνομα-κλάσης(λίστα παραμέτρων)
```

Το *όνομα-κλάσης* είναι το όνομα της κλάσης από την οποία θέλουμε να δημιουργήσουμε ένα αντικείμενο. Το όνομα της κλάσης ακολουθούμενο από τις παρενθέσεις προσδιορίζει τον κατασκευαστή της κλάσης. Όπως αναφέραμε, αν στην κλάση δεν ορίζεται κανένας κατασκευαστής, η C# θα δημιουργήσει έναν χωρίς παραμέτρους αυτόματα και θα χρησιμοποιήσει αυτόν. Επομένως ο τελεστής new μπορεί να χρησιμοποιηθεί για τη δημιουργία αντικειμένων οποιασδήποτε κλάσης. Η παραπάνω έκφραση επιστρέφει μια αναφορά προς τις δεσμευμένες θέσεις στη μνήμη που περιέχουν τις τιμές των πεδίων, έτσι όπως ορίστηκαν από τον κατασκευαστή. Η δήλωση μιας μεταβλητής τύπου αναφοράς και η αρχικοποίησή της (δηλαδή η δημιουργία του αντικειμένου) μπορούν προφανώς να γίνουν σε ξεχωριστά μέρη στον κώδικα. Σε αυτήν την περίπτωση όμως θα πρέπει να προσέχουμε την ορατότητα της μεταβλητής, όπως στο παρακάτω παράδειγμα:

```
class A { }   //δήλωση δυο απλών κλάσεων;
class B { }   //παρατηρούμε ότι δεν ορίζουν κατασκευαστές
class C
```

```
{
   A aObject; //δήλωση ενός αντικειμένου που είναι ορατό σε όλες τις μεθόδους
              //της κλάσης C
   void Initialize()
   {
      B bObject;
      aObject = new A(); //το aObject είναι ορατό
                         // και η αρχικοποίηση γίνεται εδώ
   }
   void DoNothing()
   {
      bObject = new B(); //ERROR! το bObject δεν είναι ορατό μόνο στην
                         //Initialize και δεν μπορεί να αρχικοποιηθεί από εδώ!
   }
}
```

4.3.4. Αρχικοποιητές

Παραπάνω είδαμε ότι μια έκφραση δημιουργίας αντικειμένων αποτελείται από τη λέξη new ακολουθούμενη από τον κατασκευαστή της κλάσης και τη λίστα παραμέτρων του. Ένας *αρχικοποιητής αντικειμένων(object initializer)* επεκτείνει το συντακτικό αυτό τοποθετώντας μια λίστα αρχικοποιήσεων μελών στο τέλος της έκφρασης. Έτσι επιτρέπεται η ανάθεση τιμών στα πεδία και τις ιδιότητες όταν δημιουργείται ένα νέο αντικείμενο.

Το συντακτικό έχει δυο μορφές και φαίνεται παρακάτω. Η πρώτη μορφή περιλαμβάνει τη λίστα παραμέτρων του κατασκευαστή, ενώ η δεύτερη όχι. Στη δεύτερη παρατηρούμε ότι δεν χρειάζεται να χρησιμοποιήσουμε τις παρενθέσεις του κατασκευαστή:

```
new TypeName(ArgList) { FieldOrProp = InitExpr, FieldOrProp = InitExpr, ...}
new TypeName { FieldOrProp = InitExpr, FieldOrProp = InitExpr, ...}
```

Για παράδειγμα, για μια κλάση με δυο public ακέραια πεδία x και y, μπορούμε να χρησιμοποιήσουμε την ακόλουθη έκφραση για τη δημιουργία ενός αντικειμένου:

```
new Point{ x = 5, y = 6 };
```

Τα σημεία που πρέπει να θυμώμαστε για τους αρχικοποιητές είναι τα ακόλουθα:

- Τα πεδία και οι ιδιότητες που αρχικοποιούνται πρέπει να είναι προσβάσιμα στον κώδικα που δημιουργεί το αντικείμενο.
- Η αρχικοποίηση πραγματοποιείται μετά το πέρας της εκτέλεσης του κατασκευαστή, οπότε μπορεί ο κατασκευαστής να έχει αρχικοποιήσει με έναν τρόπο τα πεδία, αλλά τελικά θα έχουν τις τιμές που ορίζει ο αρχικοποιητής.

4.4. Καταστροφή αντικειμένων

Σε μερικές γλώσσες όπως η C++, ο προγραμματιστής χρειάζεται να δεσμεύσει ρητά τους πόρους του συστήματος που χρειάζεται ένα αντικείμενο, πριν αυτό χρησιμοποιηθεί. Το αντίστροφο συμβαίνει και όταν πρέπει ένα αντικείμενο να καταστραφεί. Η αποδέσμευση των πόρων που χρησιμοποιούσε ένα αντικείμενο πρέπει να γίνει ρητά από τον προγραμματιστή. Αν οι πόροι δεν αποδεσμεύονται (ειδικά αν αυτοί είναι μνήμη) τότε λέμε ότι συμβαίνει *διαρροή μνήμης (memory leak)* και το πρόγραμμα δεσμεύει όλο και περισσότερους πόρους. Από την άλλη μεριά, αν οι πόροι αποδεσμευτούν νωρίς, μπορεί να συμβεί απώλεια δεδομένων, αλλοίωση περιοχών στη μνήμη και μπορεί να προκύψουν null pointer exceptions.

Αυτοί οι κίνδυνοι δεν υπάρχουν στο .NET. Ο κύκλος ζωής ενός αντικειμένου είναι αποκλειστική δουλειά του CLR μέσα από το μηχανισμό της *συλλογής σκουπιδιών (garbage collection).* Ο συλλογέας σκουπιδιών ελέγχει περιοδικά το σωρό (heap) στη μνήμη για αντικείμενα που πλέον δεν χρειάζονται στο πρόγραμμα – για την ακρίβεια που δεν αναφέρονται από κανένα άλλο αντικείμενο και αποδεσμεύει αυτόματα τους πόρους που δεσμεύουν τα αντικείμενα αυτά.

4.4.1. Καταστροφείς

Οι καταστροφείς (destructors) επιτελούν τις απαραίτητες ενέργειες για το «καθάρισμα» ή την αποδέσμευση πόρων, όπως για παράδειγμα μια σύνδεση στη βάση δεδομένων ή το δίκτυο, όταν ένα αντικείμενο μιας κλάσης δεν αναφέρεται άλλο από οπουδήποτε στον κώδικα. Τα σημεία που πρέπει να θυμόμαστε για τους καταστροφείς είναι τα ακόλουθα:

- Μπορεί να υπάρχει μόνο ένας καταστροφέας ανά κλάση.
- Ένας καταστροφέας δεν μπορεί να έχει παραμέτρους.
- Ένας καταστροφέας έχει το ίδιο όνομα με την κλάση αλλά από αυτό προηγείται μια πεσπιρωμένη (tilde) ~.
- Ένας καταστροφέας ενεργεί πάνω σε αντικείμενα μιας κλάσης και κατά συνέπεια δεν υπάρχουν στατικοί κατασκευαστές.
- *Δεν γίνεται να κληθεί ένας καταστροφέας από τον κώδικα.* Αντίθετα καλείται αυτόματα κατά τη διαδικασία της συλλογής σκουπιδιών, όταν ο συλλογέας αναλύσει τον κώδικα και αποφασίσει ότι δεν υπάρχει κάποιο μονοπάτι εκτέλεσης που να αναφέρεται στο αντικείμενο. Δεν υπάρχει τρόπος να γνωρίζουμε πότε θα κληθεί ο καταστροφέας.

Η δήλωση ενός καταστροφέα μοιάζει με την παρακάτω:

```
MyClass
{
   ~MyClass() // destructor
   {
      CleanupCode
   }
   ...
}
```

Δεν χρειάζεται να υλοποιήσουμε έναν καταστροφέα αν δεν τον χρειαζόμαστε πραγματικά γιατί εισάγει κόστος στην επίδοση. Ένας καταστροφέας πρέπει να αποδεσμεύει μόνο εξωτερικούς πόρους που διαχειρίζεται ένα αντικείμενο. Ένας καταστροφέας δεν πρέπει να χρησιμοποιεί άλλα αντικείμενα γιατί δεν μπορούμε να ξέρουμε αν αυτά έχουν ήδη καταστραφεί.

Αν ο κώδικας μας περιέχει μη διαχειρισμένους πόρους, δηλαδή πόρους που δεν διαχειρίζεται το .NET, και οι πόροι αυτοί πρέπει να αποδεσμευτούν, η χρήση ενός καταστροφέα δεν είναι ενδεδειγμένη, γιατί δεν ξέρουμε πότε θα εκτελεστεί. Στην περίπτωση αυτή χρησιμοποιείται μια πιο προχωρημένη μέθοδος που δεν θα εξετάσουμε: η χρήση κλάσεων που υλοποιούν το interface IDisposable.

4.5. Στατικά μέλη και κλάσεις

Μέχρι στιγμής εξετάσαμε τα πιθανά μέλη μιας κλάσης τα οποία είναι γνωστά ως *μέλη στιγμιότυπου (instance members).* Όπως είδαμε στην παράγραφο 4.2.3, για να προσπελαστούν εκτός της κλάσης στην οποία ορίζονται, πρέπει να δημιουργήσουμε ένα αντικείμενο της συγκεκριμένης κλάσης. Αυτό σημαίνει ότι ουσιαστικά τα μέλη αυτά ανήκουν σε κάθε αντικείμενο και όχι στην ίδια την κλάση. Αυτό φαίνεται και στο παρακάτω παράδειγμα:

```
class MyClass
{
   public int a;
   public MyClass(int p){a = p};
}
class Program
{
   static void Main(string[] args)
   {
      //δήλωση και αρχικοποίηση δυο διαφορετικών αντικειμένων
      MyClass ob1 = new MyClass(1);// το πεδίο a του αντικειμένου έχει τιμή 1
      MyClass ob2 = new MyClass(2);// το πεδίο a του αντικειμένου έχει τιμή 2
   }
}
```

4.5.1. Στατικά μέλη

Εκτός των πεδίων στιγμιότυπων οι κλάσεις μπορούν να έχουν και τα *στατικά μέλη (static members).* Καλούμε ως στατικά τα μέλη που ανήκουν στην ίδια την κλάση (στον ίδιο τον τύπο δεδομένων) αντί να ανήκουν σε κάθε αντικείμενο. Ορίζουμε στατικά μέλη ακριβώς με τον ίδιο τρόπο που ορίζουμε μέλη στιγμιότυπου, προσθέτοντας πριν το όνομα του μέλους τον τροποποιητή **static**. Τα στατικά μέλη μιας κλάσης είναι κοινά σε όλα τα αντικείμενα της κλάσης αλλά και όλα τα υπόλοιπα αντικείμενα (ανάλογα με την προσβασιμότητά τους).

Ο χρόνος ύπαρξης των στατικών μελών είναι διαφορετικός από αυτόν των μελών στιγμιότυπου. Τα μέλη στιγμιότυπου αρχικοποιούνται κατά τη δημιουργία του στιγμιότυπου και παύουν να υπάρχουν όταν αυτό καταστρέφεται. Αντίθετα τα στατικά μέλη υπάρχουν και είναι προσπελάσιμα ακόμα και αν δεν υπάρχουν στιγμιότυπα της κλάσης όπου περιέχονται. Αν κάποιο στατικό πεδίο αρχικοποιείται κατά τη δήλωσή του, τότε η αρχικοποίηση συμβαίνει πριν τη χρήση οποιουδήποτε στατικού πεδίου της κλάσης όπου ανήκει, αλλά όχι απαραίτητα κατά το ξεκίνημα της εκτέλεσης του προγράμματος.

Από τη λίστα των πιθανών μελών μιας κλάσης δύο μέλη δεν μπορούν να δηλωθούν ως στατικά: Οι σταθερές (constants) και οι indexers. Οι σταθερές έχουν αυτή την ιδιομορφία γιατί έχουν παρόμοια συμπεριφορά με ένα static πεδίο: είναι ορατές σε κάθε στιγμιότυπο μιας κλάσης και είναι διαθέσιμες ακόμα και αν δεν υπάρχουν στιγμιότυπα της κλάσης. Αντίθετα με τα πραγματικά στατικά πεδία όμως, οι σταθερές δεν έχουν μια ξεχωριστή δική τους θέση στη μνήμη και μεταφέρονται στη μνήμη από τον μεταγλωττιστή κατά την εκτέλεση του προγράμματος.

Στατικά πεδία

Στο παρακάτω παράδειγμα αντιπαραθέτουμε ένα στατικό με ένα μέλος στιγμιότυπου και δημιουργούμε δυο αντικείμενα για να δούμε τη διαφορά:

```
public class Student
{
   public string Name; // Instance field
   public static int Population; // Static field
   public Student(string n) // Constructor
   {
      Name = n; // ανάθεση τιμής στο πεδίο στιγμιότυπου
      Population = Population + 1; // αύξηση κατά ένα στον πληθυσμό
   }
}
class Program
{
   static void Main()
   {
      Student p1 = new Student("Nick");
      Student p2 = new Student("George");
      Console.WriteLine (p1.Name); // Nick
      Console.WriteLine (p2.Name); // George
      Console.WriteLine (Student.Population); // 2
   }
}
```

Όπως παρατηρούμε στο παραπάνω παράδειγμα χρησιμοποιούμε ένα στατικό πεδίο χρησιμοποιώντας το όνομα της κλάσης και όχι το όνομα κάποιου αντικειμένου.

Στατικές μέθοδοι

Εκτός από τα στατικά πεδία, υπάρχουν και οι στατικές μέθοδοι. Οι στατικές μέθοδοι, όπως όλα τα στατικά μέλη, σχετίζονται με την ίδια την κλάση. Ωστόσο με τις στατικές μεθόδους μπορούμε να προσπελάσουμε *μόνο* στατικά μέλη, δηλαδή στατικά πεδία, στατικές ιδιότητες και άλλες στατικές μεθόδους. Για παράδειγμα, η ακόλουθη κλάση περιέχει ένα στατικό πεδίο και μια στατική μέθοδο. Παρατηρούμε, ότι το σώμα της στατικής μεθόδου προσπελαύνει ένα στατικό πεδίο:

```
class MyClass
{
   static public int A; // Static field
```

```
   static public void PrintValA() // Static method
   {
      Console.WriteLine("Value of A: {0}", A);
   }
}

class Program
{
   static void Main()
   {
      MyClass.A = 10; // Use dot-syntax notation
      MyClass.PrintValA(); // Use dot-syntax notation
   }
}
```

Στατικοί κατασκευαστές

Οι κατασκευαστές μιας κλάσης μπορούν επίσης να είναι στατικοί. Ενώ ένας κανονικός κατασκευαστής, αρχικοποιεί κάθε νέο αντικείμενο μιας κλάσης, ένας στατικός κατασκευαστής αρχικοποιεί τα στατικά πεδία μιας κλάσης. Οι στατικοί κατασκευαστές μοιάζουν με τους κανονικούς κατασκευαστές, αφού το όνομα ενός στατικού κατασκευαστή πρέπει να είναι το ίδιο με αυτό της κλάσης, ενώ δεν μπορεί να επιστρέφει κάποια τιμή. Αντίθετα οι στατικοί κατασκευαστές, διαφέρουν από τους κατασκευαστές στιγμιότυπου, αφού έχουν τη λέξη static στη δήλωσή τους, μπορεί να υπάρχει μόνο ένας στατικός κατασκευαστής, ενώ μπορούν να υπάρχουν πολλαπλοί κατασκευαστές στιγμιότυπου και τέλος οι στατικοί κατασκευαστές δεν μπορούν έχουν τροποποιητές πρόσβασης. Ένας στατικός κατασκευαστής θα έμοιαζε με τον παρακάτω:

```
class Class1
{
   static Class1()
{
   ... // Do all the static initializations.
}
...
```

Άλλα σημαντικά σημεία που πρέπει να θυμόμαστε για τους στατικούς κατασκευαστές είναι τα εξής:

- Μια κλάση μπορεί να έχει ταυτόχρονα στατικό κατασκευαστή και κατασκευαστή στιγμιότυπου.
- Όπως οι στατικές μέθοδοι, ένας στατικός κατασκευαστής δεν μπορεί να προσπελάσει μέλη στιγμιότυπου και δεν μπορεί να χρησιμοποιήσει το this.
- Δεν μπορούμε να καλέσουμε έναν στατικό κατασκευαστή από το πρόγραμμά μας. Καλούνται αυτόματα από το σύστημα κάποια στιγμή:
  - Πριν τη δημιουργία κάποιου αντικειμένου της κλάσης.
  - Πριν την αναφορά σε οποιοδήποτε στατικό μέλος της κλάσης.

Ακολουθεί ένα παράδειγμα με στατικό κατασκευαστή:

```
class RandomNumberClass
{
   private static Random RandomKey; // Private static field
   static RandomNumberClass() // Static constructor
   {
      RandomKey = new Random(); // Αρχικοποίηση του RandomKey
   }
   public int GetRandomNumber()
   {
      return RandomKey.Next();
   }
}
class Program
{
   static void Main()
   {
      RandomNumberClass a = new RandomNumberClass();
      RandomNumberClass b = new RandomNumberClass();
      Console.WriteLine("Next Random #: {0}", a.GetRandomNumber());
```

```
        Console.WriteLine("Next Random #: {0}", b.GetRandomNumber());
    }
}
```

Στο παραπάνω παράδειγμα χρησιμοποιείται η κλάση Random που παρέχεται από τη βιβλιοθήκη κλάσεων του .NET για να παραχθούν τυχαίοι αριθμοί.

4.5.2. Στατικές κλάσεις

Πέραν των στατικών μελών μπορούμε να ορίσουμε και *στατικές κλάσεις (static classes)*. Μια στατική κλάση είναι μια κλάση που περιέχει μόνο στατικά μέλη. Οι στατικές κλάσεις ορίζονται τοποθετώντας τον τροποποιητή static μπροστά από τη λέξη class. Τέτοιες κλάσεις χρησιμεύουν για την ομαδοποίηση δεδομένων και μεθόδων που δεν επηρεάζονται από δεδομένα στιγμιότυπων. Οι στατικές κλάσεις χρησιμοποιούνται συχνά για την ομαδοποίηση πεδίων και μεθόδων η χρήση των οποίων είναι κοινή σε όλες τις υπόλοιπες κλάσεις, όπως για παράδειγμα μια βιβλιοθήκη μαθηματικών πράξεων.

Τα σημεία που πρέπει να θυμόμαστε για τις στατικές κλάσεις είναι τα ακόλουθα:

- Η ίδια η κλάση πρέπει να δηλωθεί ως static.
- Όλα τα μέλη της κλάσης πρέπει να είναι static.
- Η στατική κλάση μπορεί να έχει ένα στατικό κατασκευαστή, αλλά δεν μπορεί να έχει έναν κατασκευαστή στιγμιότυπου.
- Οι στατικές κλάσεις δεν μπορούν να κληρονομηθούν.

Χρησιμοποιούμε τις στατικές κλάσεις ακριβώς όπως θα χρησιμοποιούσαμε οποιοδήποτε στατικό μέλος (ακόμα και αν δεν ήταν στατική η κλάση), χρησιμοποιώντας το όνομα της κλάσης και το όνομα του μέλους. Ακολουθεί ένα παράδειγμα:

```
static public class MyMath
{
   public static float PI = 3.14f;
   public static bool IsOdd(int x){ return x % 2 == 1; }
   public static int Times2(int x){ return 2 * x; }
}
class Program
{
   static void Main( )
   {
      int val = 3;
      Console.WriteLine("{0} is odd is {1}.", val, MyMath.IsOdd(val));
      Console.WriteLine("{0} * 2 = {1}.", val, MyMath.Times2(val));
   }
}
```