






Modulul 2

C# - Noțiuni generale

Overview

- ▶ Structura unui program C#
- ▶ Operațiuni Intrare/ieșire de bază
- ▶ Sistemul de tipuri
- ▶ Practici recomandate



25.10.2014
2


În acest modul, se va discuta structura de bază a unui program C#, analizând un exemplu care funcționează. Se va folosi clasa Console pentru interacțiune de bază cu utilizatorul.

Se va studia și sistemul de tipuri, trecând în revistă tipurile valoare și tipurile referință. Vom vedea în ce categorie intră tipurile „int”, „float” sau „string”.

De asemenea, se va analiza tratarea erorilor și documentarea codului. În ultima parte se va discuta despre modul în care codul se compilează, rulează și cum se face debugging.

La finalul acestui modul, studenții vor putea să:

- Explice structura unui program simplu C#
- Folosească clasa Console pentru operațiuni Intrare/ieșire de bază
- Să identifice dacă un tip este valoare sau referință
- Să folosească operatorii în mod corespunzător
- Compileze și execute un program C#
- Folosească debugger-ul pentru a urmări execuția unui program





Structura unui program C#

1. Programul "Hello World!"
2. Clasele
3. Metoda Main
4. Directiva *using* și *System Namespace*
5. Demonstrație în Visual Studio

25.10.2014

3

În această lecție, veți învăța structura de bază a unui program C#. Veți analiza un program simplu ce conține toate elementele de bază. Veți învăța, de asemenea, cum să folosiți Microsoft Visual Studio pentru a crea și edita un program C#.



I. Programul "Hello World!"

```
using System;  
class Hello  
{  
    public static void Main()  
    {  
        Console.WriteLine("Hello, World");  
    }  
}
```

25.10.2014

4

Primul program pe care majoritatea oamenilor îl scriu când învață un limbaj de programare este programul "Hello World!". În continuare, vom analiza versiunea C# a celebrului prim program.

Codul exemplu din slide conține toate elementele esențiale pentru un program C# și este simplu de testat. În urma rulării, la consola va apărea textul: *Hello, World*.

În continuare, vom analiza acest program simplu pentru a învăța mai multe despre elementele de bază ale unui program C#.

2. Clasele

- ▶ O aplicație C# este o colecție de clase, structuri și tipuri
- ▶ O clasă este o colecție de date și metode
- ▶ Sintaxa:

```
class name  
{  
    .....  
}
```



- ▶ O aplicație C# poate fi structurată în mai multe fișiere

25.10.2014

5

În C#, o aplicație este o colecție de una sau mai multe clase, structuri de date și alte tipuri. În acest modul, clasa este definită ca un set de date combinate cu metode (funcții) de manipulare a acestor date. Într-un modul viitor, se va discuta mai pe larg despre clase.

Aruncând o privire asupra programului *Hello, World* se observă existența unei singure clase, numite *Hello*. Această clasă este introdusă prin cuvântul cheie *class*. Urmează apoi deschiderea unei acolade {. Tot ce urmează până la acolada închisă } aparține clasei.

3. Metoda Main

- ▶ Când se scrie metoda Main, trebuie să:
 - ▶ 'M'-ul de început trebuie să fie mare: "Main"
 - ▶ Funcția Main este intrarea în program
 - ▶ Funcția Main trebuie declarată ca funcție statică, publică, ce nu întoarce nicio valoare

```
public static void Main()
{
    ....
}
```

- ▶ Mai multe clase pot avea funcția Main
- ▶ La ieșirea din funcția Main, aplicația se termină.

25.10.2014
6



Orice aplicație trebuie să aibe un început. Când o aplicație C# rulează, ea începe la metoda numită Main.

Important: Limbajul C# este case-sensitive. Numele metodei trebuie să fie întotdeauna "Main", scris cu 'M'-mare, restul fiind litere mici.

Deși pot exista multe clase într-o aplicație C#, poate exista un singur punct de început. Este posibil ca mai multe clase să aibe definite metode Main, dar doar una dintre acestea va fi executată. Va trebui ca programatorul să specifice care dintre metodele Main să fie executate. Se va discuta despre acest lucru în secțiunea de compilare.

Forma în care este definită metoda **Main** este de asemenea importantă. Forma standard este definirea ei ca "**static void**". Se va discuta pe larg într-un modul viitor ce înseamnă acestea.

Aplicația va rula până la ieșirea din funcția Main, fie prin terminarea ei, fie prin întâlnirea unei operații de "return".

4. Directiva *using* și *System Namespace*

- ▶ Platforma .NET pune la dispoziția programatorului o colecție de clase
 - ▶ Colecțiile de clase poartă numele de “**namespace**”
- ▶ Clasele se accesează cu ajutorul namespace-ului

```
System.Console.WriteLine("Hello, World");
```

- ▶ Folosind directiva *using*, nu mai trebuie specificat namespace-ul unei clase

```
using System;
....
Console.WriteLine("Hello, World");
```

25.10.2014
7

Ca parte a platformei .NET, C# are la dispoziție o colecție foarte mare de clase care se ocupă cu o serie mare de operațiuni. Pentru a organiza acest număr foarte mare de clase, se folosesc Namespace-uri. Un namespace este un grup de clase ce au legătură una cu alta. Un namespace poate conține și alte namespace-uri, formând o structură arborescentă.

Platforma .NET este alcătuită din multe namespace-uri, cel mai important dintre acestea fiind **System**. Namespace-ul System conține clasele pe care majoritatea aplicațiilor le folosesc pentru integrarea cu sistemul de operare. Cele mai des folosite clase se ocupă de operațiuni Intrare/Ieșire. Cum se întâmplă și cu alte limbaje de programare, C# nu are metode de I/E proprii, depinzând de sistemul de operare pentru a oferi o interfață compatibilă pentru operațiunile de intrare/ieșire.

O clasă dintr-un anumit namespace poate fi referită prefixând namespace-ul la numele clasei. De exemplu, clasa Console este conținută în namespace-ul System. Pentru a apela metoda WriteLine a clasei Console, se poate scrie următoarea linie de cod:

```
System.Console.WriteLine("Hello, World");
```




5. Demonstrație în Visual Studio

- ▶ Crearea unui proiect Console Application în C#
 - ▶ Accesați meniul File => New => Project
 - ▶ Din partea stânga selectați Visual C# (Other Languages)
 - ▶ Selectați template-ul Console Application
- ▶ Compilarea și rularea
 - ▶ Compilare – meniul Build => Build Solution (tasta F6)
 - ▶ Rulare – meniul Debug => Start Debugging (tasta F5)
- ▶ Accesarea ferestrelor utile în Visual Studio
 - ▶ Folosiți meniul View



25.10.2014
8

Exista mai multe modalități de a rula un program din Visual Studio:

- Start Debugging => permite utilizarea mecanismelor specifice de depanare în timpul execuției programului
- Start Without Debugging => lansează aplicația fără să țină cont de informațiile de depanare prezente în cadrul proiectului

Printre cele mai utilizate ferestre in Visual Studio se găsesc următoarele:

- Solution Explorer => conține structura de fișere a proiectului deschis
- Error List => reprezintă o listă de erori și avertismente obținute la compilare
- Properties Window => folosit pentru a afla mai multe informații despre un element atunci când acesta este selectat în cadrul unui designer grafic
- Class View => analizează suplimentar conținutul fișierelor pentru a crea o structură detaliată de clase și metode existente în cadrul proiectului





Operațiuni Intrare/ieșire de bază

1. Clasa Console
2. Metodele Write și WriteLine
3. Metodele Read și ReadLine

25.10.2014

9

În continuare, se va discuta despre modul în care se fac operațiile de Intrare/ieșire în C# folosind clasa **Console**. Veți învăța atât cum se afișază date folosind metodele **Write** și **WriteLine**, cât și cum se citesc date introduse de utilizator cu ajutorul metodelor **Read** și **ReadLine**.

I. Clasa Console

- ▶ Oferă acces la streamurile de standard input, standard output și standard error
- ▶ Are sens doar pentru aplicații pentru consolă
 - ▶ Standard input – tastatură
 - ▶ Standard output – monitor
 - ▶ Standard error – monitor
- ▶ Toate aceste streamuri pot fi redirectate

25.10.2014
10

Clasa Console oferă aplicațiilor acces la streamurile de intrare, ieșire și eroare.

Standard input este de obicei tastatura, orice tasta introdusă de utilizator putând fi citită de la input stream. Asemănător, standard output este de obicei ecranul calculatorului, la fel fiind și în cazul standard error.

Important: Aceste streamuri ca și clasa Console au sens doar pentru Console Applications, aplicații ce sunt făcute să ruleze într-o fereastră **Command**.

Oricare din aceste 3 streamuri poate fi redirectat către alt dispozitiv sau către un fișier. Acest lucru se poate face fie programatic, fie de către utilizator, la pornirea aplicației.

2. Metodele **Write** și **WriteLine**

- ▶ **Console.Write** și **Console.WriteLine** afișează informații către streamul de output
 - ▶ **WriteLine** adaugă și caracterul pentru linie nouă la final
- ▶ Ambele metode sunt supraîncărcate
 - ▶ Există mai multe modalități de a fi utilizate
- ▶ Argumentele pe care le pot primi includ stringuri formate și parametri

25.10.2014

11

Metodele `Console.Write` și `Console.WriteLine` sunt folosite pentru a afișa informații la output stream. Cele două metode sunt foarte asemănătoare între ele, singura diferență fiind că `WriteLine` adaugă caracterul de linie nouă la final.

Ambele metode sunt supraîncărcate[1]. Le puteți apela cu număr și tip de parametri diferiți. De exemplu, următoarea linie de cod va afișa "99" pe ecran:

```
Console.WriteLine(99);
```



Următoarea bucată de cod va afișa pe ecran "Hello, World":

```
Console.WriteLine("Hello, World");
```

Formatarea textului

Există variante mai puternice ale metodelor `Write` și `WriteLine`, care primesc ca argumente un string formatat și parametri adiționali. Stringul formatat specifică cum să fie afișate datele și conține marcare speciale care vor fi apoi înlocuite în ordine de către parametri ce urmează. De exemplu, următorul cod va afișa mesajul "Suma dintre 100 și 130 este 230":

```
Console.WriteLine("Suma dintre {0} si {1} este {2}", 100, 130, 100+130);
```



3. Metodele **Read** și **ReadLine**

- ▶ **Console.Read** și **Console.ReadLine** citesc informații introduse de utilizator
 - ▶ **Read** citește următorul caracter
 - ▶ **ReadLine** citește următoarea linie
- ▶ Rezultatul întors este în format text

25.10.2014 12

Se pot obține date de la utilizator din linia de consolă folosind metodele **Console.Read** și **Console.ReadLine**.

Metoda **Console.Read**

Metoda va citi primul caracter introdus la tastatură. În caz de eroare, va întoarce -1. Altfel, va întoarce un int reprezentând caracterul respectiv.

Metoda **Console.ReadLine**

Această metodă citește toate caracterele până la sfârșitul liniei (caracterul carriage return). Rezultatul este un string de caractere.

Sistemul de tipuri

Sistemul de tipuri are următoarea clasificare

Tipuri valoare

- Variabila conține direct valoarea
- Exemple: **char, int**

```
int valType;
valType = 42;
```

42

Tipuri referință

- Variabila conține doar o referință către locul unde sunt reținute datele
- Datele sunt reținute într-o zonă separată de memorie

```
string refType;
refType = "Hello"
```

• → Hello



➤ Când definești o variabilă trebuie să alegi tipul de date care se potrivește cel mai bine cu cerințele. Tipul de date determină care sunt valorile permise pentru acea variabilă, implicit care sunt operațiile care sunt permise pentru acea variabilă. C# suporta tipuri de bază, cum ar fi **int**, **long** și **bool**. Aceste tipuri mai sunt numite și tipuri valoare. C# suporta de asemenea și tipuri mai complexe și puternice, numite tipuri referință.

Tipuri valoare

Variabilele tip valoare sunt tipurile de bază suportate de limbaj, precum **char** sau **int**. Ele reprezintă cele mai simple tipuri din C#. Variabilele de acest tip conțin direct datele în variabilă.

Tipuri referință

Variabilele de tip referință conțin doar o referință cu privire la zona unde sunt stocate datele și nu datele direct. Ați avut deja de-a face cu astfel de tipuri de variabile. Vectorii, stringurile și excepțiile sunt toate tipuri referință ce sunt implementate în compilatorul C# și în platforma .NET. Clasele, fie deja definite sau definite de utilizator, sunt de asemenea variabile tip referință.

Tipuri valoare predefinite/definite de utilizator

Tipuri predefinite

- ▶ int
- ▶ float
- ▶ double
- ▶ bool
- ▶ char

Tipuri definite de utilizator

- ▶ enum
- ▶ struct


Atentie! Tipul **string** este o valoare de tip referință



▪ **Tipurile valoare** includ: tipurile predefinite și tipurile definite de utilizator. Diferența dintre cele două este minimală având în vedere că variabilele indiferent de tip sunt folosite în același fel. Toate tipurile predefinite care sunt de fapt tipurile valoare conțin date ceea ce înseamnă că nu pot avea valoarea **null**.

➤ Tipurile predefinite mai sunt numite tipuri de bază sau tipuri simple. Acestea sunt identificate prin cuvinte cheie rezervate. Cuvintele cheie nu reprezintă altceva decât aliasuri pentru structuri de date predefinite (tipurile de bază sunt definite prin aceste structuri).

▪ **byte // Reserved keyword** este echivalent cu **System.Byte // struct type**



itacademy

Microsoft .NET

Denumirea tipurilor

- ▶ Reguli
 - ▶ Folosiți litere, cifre și underscore
 - ▶ Nu folosiți cuvinte rezervate:

abstract , base, bool, default, if, finally

- ▶ Recomandări
 - ▶ Evitați să folosiți numai litere mari
 - ▶ Evitați să începeți cu “_” (underscore)
 - ▶ Evitați abrevierile
 - ▶ Utilizați regula Camel când denumirea este formată din mai multe cuvinte

25.10.2014
15

➤ Când denumiți o variabilă tineți cont de următoarele reguli și recomandări

➤ **Reguli**

- Denumirea unei variabile să înceapă cu o literă sau *underscore*
- După primul caracter să se folosească doar cifre, litere sau *underscore*
- Să nu se folosească cuvintele rezervate în denumire
- Dacă nu se vor respecta regulile de mai sus vor rezulta erori de compilare

➤ **Recomandări**



- Să se evite să se folosească numai litere mari
- Să se evite să se înceapă cu *underscore*
- Să se evite să se folosească abrevierile
- Să se folosească regula **Pascal** de denumire a variabilelor

➤ **Regula Pascal de denumire**

- În cazul în care numele variabilei este format din mai multe cuvinte prima literă din fiecare cuvânt este mare restul mici.
- Se utilizează pentru denumirea: claselor, metodelor, proprietăților, enumerațiilor, interfețelor, câmpurilor *read-only* sau constantelor, spațiilor de nume.
- Exemplu: ***void InitializeData();***

➤ **Regula Camel de denumire**

- La fel ca Pascal numai că prima literă este întotdeauna mică
- Se utilizează pentru variabilele care definesc câmpuri sau parametrii
- Exemplu: ***int loopCountMax;***

Declararea de variabile și atribuirea lor

- ▶ Specificarea tipului și a numelui variabilei


```
int    itemCount;
```
- ▶ Multiple variabile în aceeași declarație


```
int    itemCount,    numărAngajați;
```
- ▶ Asignarea valorii variabilei deja declarate


```
int    numărAngajați;
numărAngajați = 1000;
```
- ▶ Asignarea valorii la declararea variabilei


```
int    numărAngajați = 1000;
```

25.10.2014
16

➤ Variabilele care sunt declarate în metode, proprietăți sau indexări sunt numite variabile locale. Definirea unei variabile presupune specificarea tipului și numelui acesteia.

▪ Exemplu: **int itemCount;**

➤ Se pot declara mai multe variabile în aceeași linie de cod separate prin virgulă

▪ Exemplu: **int itemCount, numărAngajați;**

➤ În C# nu poți folosi variabile neinițializate. Următorul exemplu va duce la o eroare de compilare pentru că variabila afișată nu a fost inițializată:

```
int loopCount;
Console.WriteLine("{0}", loopCount);
```

➤ Pentru a atribui o valoare unei variabile se folosesc operatorii de atribuire.

➤ Următorul exemplu atribuie o valoare unei variabile care a fost declarată anterior utilizând operatorul "=":

```
int employeeNumber;
employeeNumber = 23;
```

➤ O variabilă poate fi, de asemenea, inițializată la declararea acesteia:

```
int employeeNumber = 23;
```

➤ Operatorul "=" poate fi folosit și pentru a atribui un caracter unei variabile de tip **char**:

```
char middleInitial = 'J';
```


Operatori

Operatori	Exemplu
Operatori de egalitate	== !=
Operatori relaționali	< > <= >= is
Operatori condiționali	&& ?:
Operator de incrementare	++
Operator de decrementare	--
Operatori aritmetici	+ - * / %
Operatori de atribuire	= *= /= %= += -= <<= >>= &= ^= =



- O expresie este formată din operanzi și operatori. Operatorul unei expresii definește operația care va fi aplicată asupra operanzilor.
- Există trei tipuri de operatori:
 - Unari – se aplică unui singur operand: ++, --
 - Binari – se aplică între doi operanzi: majoritatea din tabelul următor
 - Ternari – există unul singur cel condițional ?: care ia trei operatori și utilizează regula: c? x: y

Tip operator	Descriere
Operatori de atribuire	Atribuire valori unei variabile. Pentru ca operația să aibă succes valoarea din partea dreaptă a atribuirii trebuie să fie de un tip care poate fi convertit implicit la tipul variabilei din partea stângă
Operatori relaționali	Compară două valori
Operatori condiționali	Selectează între mai multe expresii în funcție de rezultatul condiției
Operatori logici	Realizează operații pe biți între două valori
Operator de incrementare	Crește valoarea operandului cu 1
Operator de decrementare	Descrește valoarea operandului cu 1
Operatori aritmetici	Realizează operații aritmetice standard

Scrieri prescurtate și precedența operatorilor

► Atribuiiri compuse

- Cele două expresii sunt echivalente

```
value = value + 3;
```

```
value += 3;
```

► Incrementare

```
value++;
```

sau

```
++value;
```

► Precedența operatorilor și asociativitatea

- Operatorii de atribuire și cel condițional (?:) sunt asociativi la dreapta

► $x = y = z \Leftrightarrow x = (y = z)$

- Restul (excepție cei amintiți anterior) sunt asociativi la stânga

► $x + y + z \Leftrightarrow (x + y) + z$

25.10.2014

18

- Atribuirea compusă funcționează pentru orice operator:

```
var += expression; // var = var + expression
```

```
var -= expression; // var = var - expression
```

```
var *= expression; // var = var * expression
```

```
var /= expression; // var = var / expression
```

```
var %= expression; // var = var % expression
```

- Operatorul de incrementare (creșterea valorii cu o unitate) este ++, iar operatorul de decrementare (valoarea inițială minus o unitate) este --.

- Operatorii de incrementare și de decrementare pot fi folosiți în două feluri:

- **Forma prefixată**, când e adăugat înaintea operandului

```
++itemCount;
```

- **Forma sufixată**, când e adăugat după operand

```
itemCount++;
```

- Diferența dintre cele două forme

- În C# o operație de atribuire returnează o valoare care este de obicei valoarea variabilei după operația de atribuire. În majoritatea cazurilor această valoare nu este luată în considerare, dar ea poate fi folosită ca în exemplul următor:

```
int itemCount = 0;
```

```
Console.WriteLine(itemCount = 2); // Prints 2
```

```
Console.WriteLine(itemCount += 40); // Prints 42
```

- La fel se întâmplă și la incrementare, respectiv decrementare. Pe lângă operația în sine, expresia va returna un rezultat care poate fi folosit sau nu. În exemplul următor rezultatul este utilizat în două atribuiri:

► Precedența operatorilor

- Când o expresie conține mai mulți operatori precedența acestora definește ordinea în care sunt evaluați
- Precedența operatorilor aritmetici este cunoscută deja din matematică (operatorii multiplicativi: *, /, și %, au prioritate mai mare decât operatorii aditivi: +, -)

► Asociativitatea

- Când un operand se află între două operații cu aceeași precedență, asociativitatea este cea care determină ordinea în care sunt realizate acestea
- De exemplu: $x+y+z$ este echivalent cu $(x+y)+z$, pe când $x=y=z$ este echivalent cu $x=(y=z)$.
 - Cu excepția operatorilor de atribuire toți ceilalți sunt asociativi la stânga
 - Operatorii de atribuire și cel condițional (?:) sunt asociativi la dreapta, adică operațiile se execută de la dreapta la stânga
- Pentru controlul execuției și a ordinii operațiilor se pot utiliza paranteze.

Clasa String

- ▶ Date în format Unicode
- ▶ Nume complet: System.String
- ▶ Un string nu se poate modifica

```
string s = "Hello";  
s[0] = 'c'; //Eroare la compilare
```

25.10.2014

19

În C#, tipul string este folosit pentru a procesa caractere Unicode multiple. Pentru comparație, tipul char este un tip valoare ce se ocupă cu un singur caracter.

Numele tipului, **string**, este prescurtare pentru clasa **System.String**. Compilatorul se ocupă de recunoașterea formei prescurtate, astfel încât oricare dintre cele două forme poate fi folosită.

Clasa **String** reprezintă un șir de caractere ce nu poate fi modificat. Metode ce par să modifice valoarea unui string întorc de fapt valoarea unei noi instanțe ce conține modificările.

Clasa **String** oferă un set foarte mare de metode. Dintre acestea, cele mai des folosite vor fi listate în acest curs.

Principalele metode, operatori și proprietăți

- ▶ Parantezele `[]`
- ▶ Metoda **Insert**
- ▶ Proprietatea **Length**
- ▶ Metoda **Copy**
- ▶ Metoda **Concat**
- ▶ Metoda **Trim**
- ▶ Metodele **ToUpper** și **ToLower**

25.10.2014

20

Parantezele `[]`

Puteți extrage un singur caracter din string folosind numele variabilei urmată de indexul caracterului între paranteze `[]`. Procesul este similar celui de la vectori. Primul caracter are indexul 0.

```
string s = "Alphabet";  
char firstchar = s[2]; // 'p'
```

Stringurile nu sunt modificabile, astfel încât un element nu poate fi modificat. Dacă se încearcă acest lucru, se va primi eroare la compilare:

```
s[2] = '*';           // Nu este valid
```

Metoda **Insert**

Această metodă este folosită pentru a insera un șir de caractere în alt șir de caractere. Primește 2 parametri, poziția de la care se va face inserția și șirul ce va fi inserat. Întoarce un nou string cu valoare specificată inserată de la poziția respectivă.

```
string s = "C is great!";  
s = s.Insert(2, "Sharp ");  
Console.WriteLine(s);           // C Sharp is great!
```

Comparații pe stringuri

► Metoda **Equals**

- Comparație de valori

► Metoda **Compare**

- Compară folosind ordinea tip dicționar
- Opțiune de case-insensitive

► Opțiuni de comparație pentru string-uri în diferite limbi

25.10.2014

22

Metoda **Equals**

Clasa `System.String` conține o metodă numită `Equals`, care poate fi folosită pentru a compara două stringuri, pentru a determina dacă sunt egale. Metoda întoarce o valoare booleană, **true** dacă cele două sunt egale și **false** altfel. Această metodă este supraîncărcată și poate fi folosită fie ca o metodă a unei instanțe, fie static:

```
string s1 = "welcome";  
string s2 = "welcome";  
if (s1.Equals(s2))  
    Console.WriteLine("The strings are the same");  
if (String.Equals(s1,s2))  
    Console.WriteLine("The strings are the same");
```

Metoda **Compare**

Se compară două stringuri din punct de vedere lexical (conform ordinei din dicționar). Valoarea întoarsă de această metodă este:

- un întreg negativ dacă primul string apare în dicționar înainte de al doilea
- 0, dacă cele două stringuri sunt egale
- un întreg pozitiv, dacă primul string apare după al doilea

```
string s1 = "Tintinnabulation";  
string s2 = "Velocipede";  
int comp = String.Compare(s1,s2);           // rezultat negativ
```

Prin definiție, un string **null** este mai mic decât orice string, chiar și stringul gol. Comparația a două stringuri **null** va întoarce 0.

Compare este supraîncărcată. Există o versiune cu 3 parametri, al treilea fiind un boolean ce specifică dacă ar trebui sau nu făcută diferența între caractere mari și caractere mici în timpul comparației.

```
s1 = "cabbage";  
s2 = "Cabbage";  
comp = String.Compare(s1, s2, true); //se ignoră diferența de caractere; se întoarce 0
```

Opțiuni de comparație pentru stringuri în diferite limbi

Metoda **Compare** este supraîncărcată pentru a da posibilitatea de a face comparația în funcție de reguli specifice diferitelor limbi. Acest lucru poate fi folosit în momentul scrierii aplicațiilor pentru piețe internaționale. Această discuție va fi reluată în următoarele cursuri din pachet.

Operatori pentru comparații pe stringuri

- ▶ Operatorii == și != sunt supraîncărcați pentru string-uri
- ▶ Sunt echivalenți cu `String.Equals` și `!String.Equals`

```
string a = "Test";  
string b = "Test";  
if (a == b) ... // Evaluată la true
```

- ▶ **Observație** Acești operatori vor compara valoarea string-urilor, nu referința!

25.10.2014

23



Operatorii == și != sunt supraîncărcați pentru clasa **String**. Astfel, următorul exemplu de corect:

```
string a = "Test";  
string b = "Test";  
if (a == b) ... // întoarce true
```

Următorii operatori și metode sunt ecivalenți:

- operatorul == este echivalent cu metoda **String.Equals**
- operatorul != este echivalent cu metoda **!String.Equals**

Ceilalți operatori relaționali (<, >, <=, >=) nu sunt supraîncărcați pentru clasa **String** și nu pot fi folosiți.





Practici recomandate

1. Comentarea aplicațiilor
2. Debugging

25.10.2014

24

În continuare, se vor discuta câteva practici recomandate când scriem aplicații C#. Se va explica modul în care este recomandat să se comenteze o aplicație, pentru a o face ușor de înțeles și întreținut / dezvoltat ulterior. Veți învăța de asemenea cum să tratați erorile care pot apărea la rulare.

I. Comentarea aplicațiilor

- ▶ Comentariile sunt importante!
 - ▶ O aplicație bine comentată permite unui dezvoltator să înțeleagă integral structura acesteia
- ▶ Comentarii pe o singură linie:


```
//Get the user's name
Console.WriteLine("What is your name? ");
name = Console.ReadLine();
```
- ▶ Comentarii pe mai multe linii:


```
/* Find the lowest number with the square root
   higher than the required value */
x = ... ;
```

25.10.2014
25



Este important să documentați toate aplicațiile pe care le scrieți. Trebuie ca nivelul comentariilor să permită unui dezvoltator care nu a fost implicat de la început în dezvoltare să înțeleagă aplicația. Comentariile utile oferă informații care nu reies ușor din cod, explicând “de ce” și nu neapărat “cum”. Dacă compania pentru care lucrați are standardele proprii privind comentariile, respectați-le.

C# oferă mai multe mecanisme pentru adăugarea comentariilor în codul aplicațiilor:

- Comentarii pe o singură linie
- Comentarii pe mai multe linii

Pentru a adăuga comentarii pe o singură linie, acestea sunt introduse de către // Până la finalul liniei, toate caracterele vor fi considerate ca făcând parte din comentariu.

Comentariile pe mai multe linii sunt introduse de către /* Până la întâlnirea caracterelor */, tot textul face parte din comentariu.

2. Debugging (depanare)

- ▶ Setarea de **Break Points** și **Watches**
 - ▶ Utile pentru a investiga execuția unui program
- ▶ Parcurgerea codului
 - ▶ Ajută la o evaluare detaliată a funcționării acestuia
- ▶ Examinarea și modificarea variabilelor
 - ▶ Oferă control sporit asupra datelor conținute de program la un moment dat

25.10.2014
26

Setarea de Break Points și Watches


Visual Studio Debugger poate fi folosit pentru introducerea unor puncte în care execuția programului este oprită pentru a vedea valoarea variabilelor.

Pentru a face acest lucru, dați click-dreapta pe o linie de cod și selectați **Insert Breakpoint**. Asemănător se și scot Breakpoints.

Fereastra Watch afișază valoarea variabilelor în timpul rulării aplicației. Variabilele ce sunt urmărite trebuie adăugate, scriindu-le numele în coloana Name. Valoarea va apărea în coloana Value. Această valoare se poate suprascrie în timpul rulării, modificând astfel din mers comportamentul programului.


Parcurgerea Codului

Odată ce aplicația s-a oprit la un Breakpoint sau chiar de la începutul rulării sale, avem opțiunea de a parcurge instrucțiunile pas cu pas, pentru a vedea comportamentul aplicației și valorile variabilelor, precum și evoluția lor. Aceste opțiuni se găsesc în meniul Debug al Visual Studio.


itacademy
your technology

Sumar

- ▶ Structura unui program C#
- ▶ Operațiuni Intrare/ieșire de bază
- ▶ Sistemul de tipuri
- ▶ Practici recomandate


Microsoft
.NET

25.10.2014

27

Intrebări:

- 1) Unde începe execuția unei aplicații C#?
- 2) Unde se termină execuția unei aplicații C#?
- 3) Câte clase poate conține o aplicație C#?
- 4) Câte metode Main poate conține o aplicație C#?
- 5) Cum se citesc date de la tastatură într-o aplicație C#?
- 6) În ce namespace se află clasa Console?