Dezvoltarea Aplicațiilor Web utilizând ASP.NET Core MVC
Curs 2

Cuprins

Introducere în C#

C# este unul dintre cele mai populare limbaje de programare, dezvoltat de Microsoft, care rulează pe arhitectura .NET.

Este folosit pentru a dezvolta o gamă variată de aplicații, de la aplicații web, mobile, desktop, până la jocuri, servicii web, VR, etc.

C# face parte din categoria limbajelor de programare orientate pe obiecte, fiind un limbaj de nivel înalt.

C++ vs C#

Aspect comparat	C++	C#
Sintaxa	Sintaxă complexă	Sintaxa este mult mai simplă, fiind de cele mai multe ori intuitivă
Tipul limbajului	Este un **limbaj de nivel mediu**, combinând limbajul low-level cu cel high-level. **Limbajul low-level** (de ex: Assembly) – codul scris este apropiat de codul mașină; oferă control asupra regiștrilor, memoriei și componentelor hardware; codul are o complexitate ridicată, necesitând cunoștințe detaliate despre arhitectura hardware. C++ combină caracteristicile low-level (ex: accesul direct la memorie și hardware) cu unele caracteristici high-level (ex: abstractizările și structurile de date complexe).	Este un **limbaj de nivel inalt** (high-level). Acestea sunt mai abstracte față de hardware, fiind **mai apropiate de limbajul natural** (uman), facilitând scrierea codului, citirea și întreținerea acestuia. Codul poate fi rulat pe diferite arhitecturi hardware, fiind portabil între diferite platforme. **Portabil** = același cod sursă poate fi rulat pe diferite platforme fără a necesita modificări semnificative sau deloc. **Platforme** însemnând combinațiile de hardware și software pe care aplicațiile sunt dezvoltate și executate: sisteme de operare (Windows, macOS, Linux), arhitecturi (x86 – PC-uri, ARM - mobile, MIPS, etc), medii de execuție (mașini virtuale, interpretoare, browsere web) și dispozitive. Exemple de limbaje: C#, Python, Java, JavaScript;

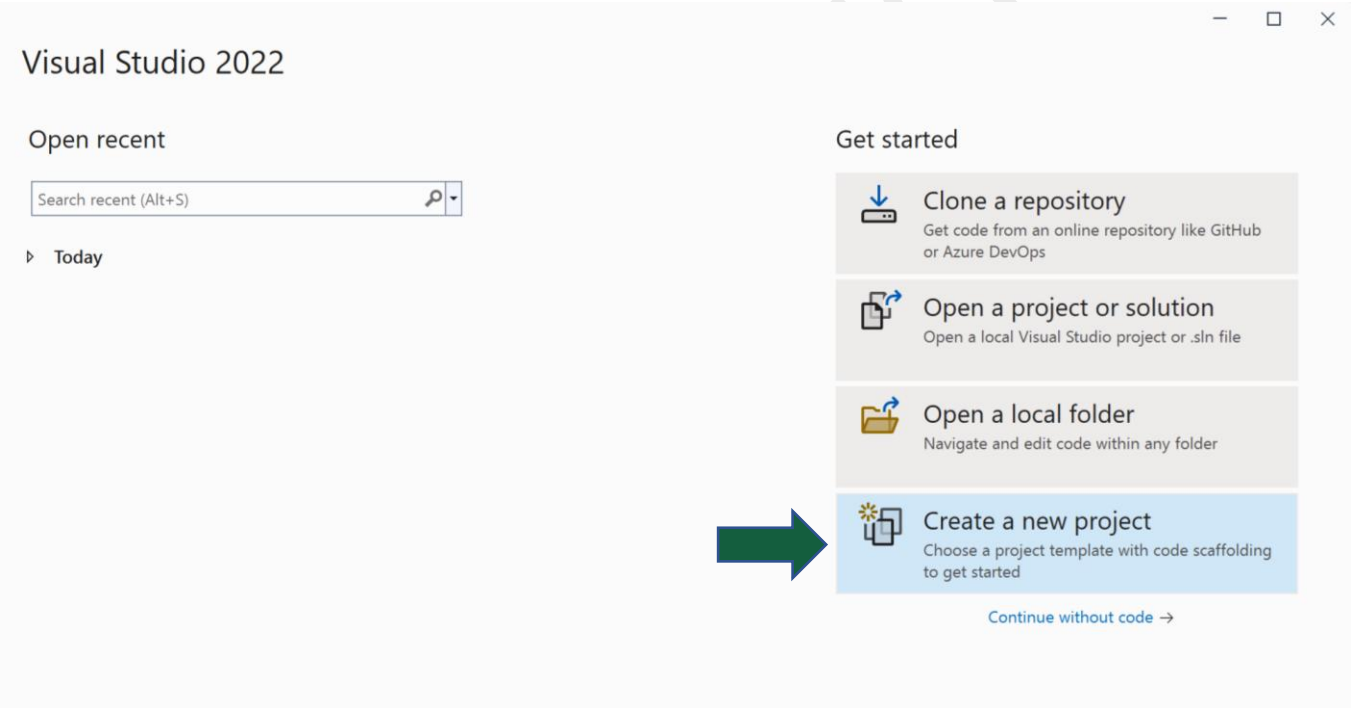
Compilare	Codul este convertit imediat după compilare în cod mașină. Codul necesită recompilare pentru diferite platforme.	Codul este compilat și convertit într-un limbaj intermediar (IL), prin intermediul componentei CLR (Common Language Runtime), cea care gestionează execuția aplicațiilor .NET. CLR încarcă assembly-ul și începe procesul de execuție. Compilatorul **JIT** (Just-In-Time) este responsabil pentru compilarea codului IL în **cod mașină**, specific platformei pe care rulează aplicația.
Gestionarea memoriei	Memoria este gestionată manual. Atunci când se creează un obiect, acesta trebuie distrus manual, eliberând astfel memoria.	Memoria este gestionată automat prin Garbage Collector, obiectele eliminându-se automat atunci când este necesar.
Moștenire	Se poate utiliza moștenirea multiplă. O clasă poate extinde mai multe clase în același timp.	Moștenirea multiplă nu este suportată în C#.
Biblioteci	**STL** (Standard Template Library) - biblioteca standard în **C++** care oferă un set de clase, funcții pentru structuri de date și algoritmi, facilitând dezvoltarea de cod eficient, reutilizabil și generic.	**Biblioteca de clase .NET (namespace-urile)** **System.Collections.Generic** – similar cu STL (listă, dicționar, coadă, stivă, etc). Funcționalitățile oferite de **LINQ**. **System.Collections** – conține colecții non-generice, bazate pe tipul **object** (ArrayList, Hashtable, etc).

Structura unui program

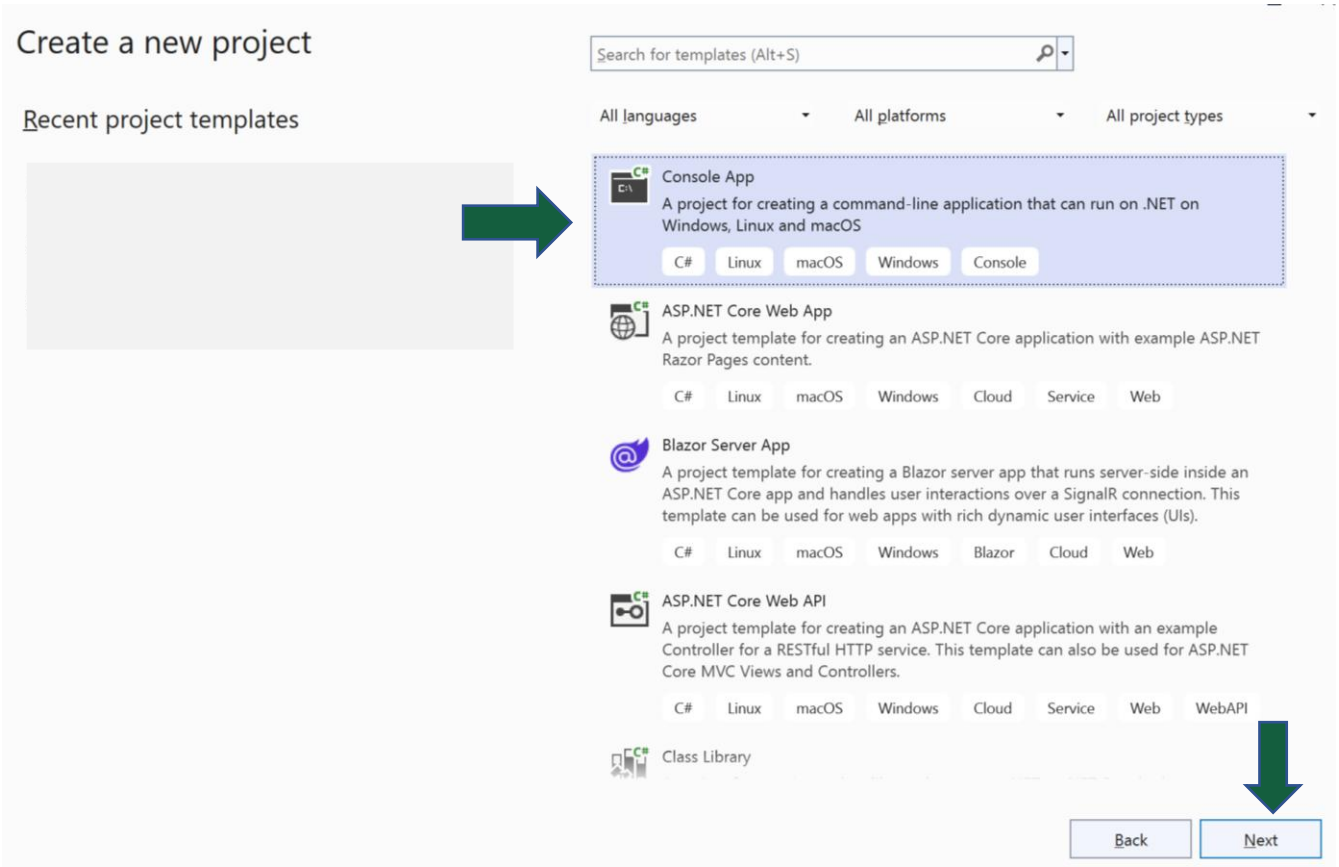
Pentru a dezvolta un program scris în C#, vom utiliza Visual Studio 2022 (**VEZI** Laborator 1 – instalare VS 2022).

Pentru început se creează un proiect folosind următoarele proprietăți:

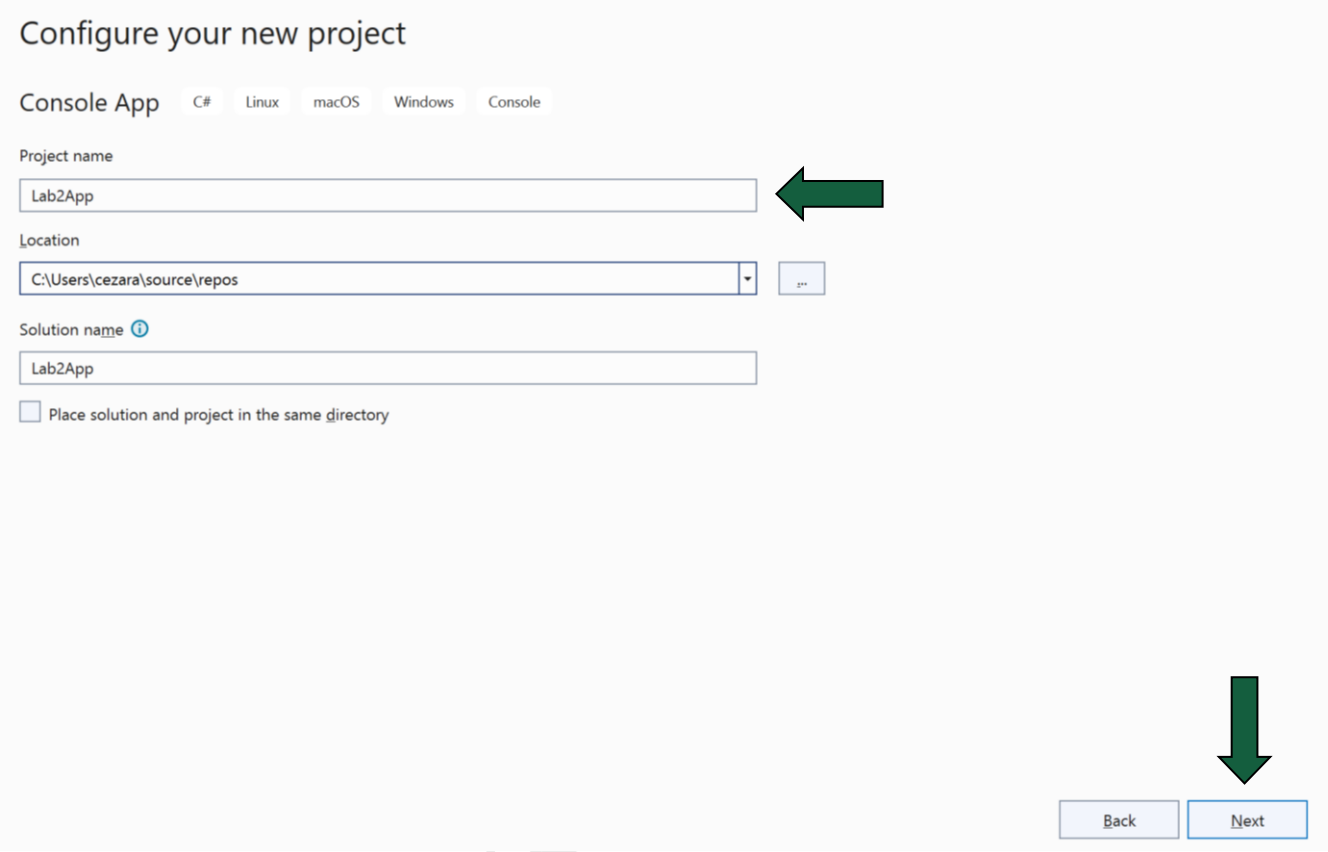
PASUL 1:



PASUL 2:

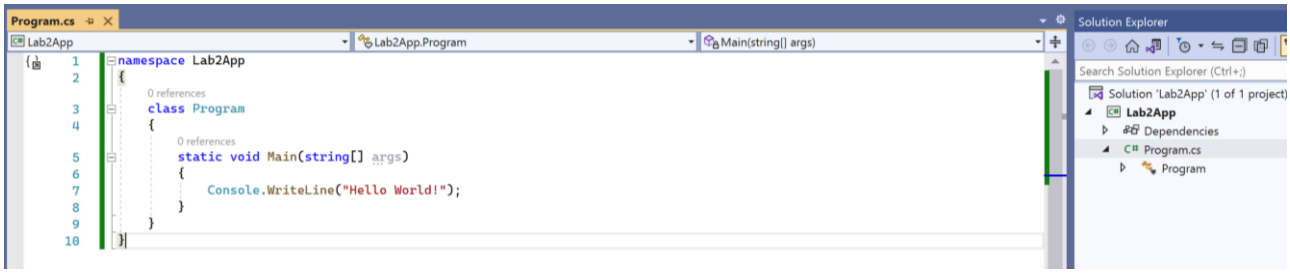


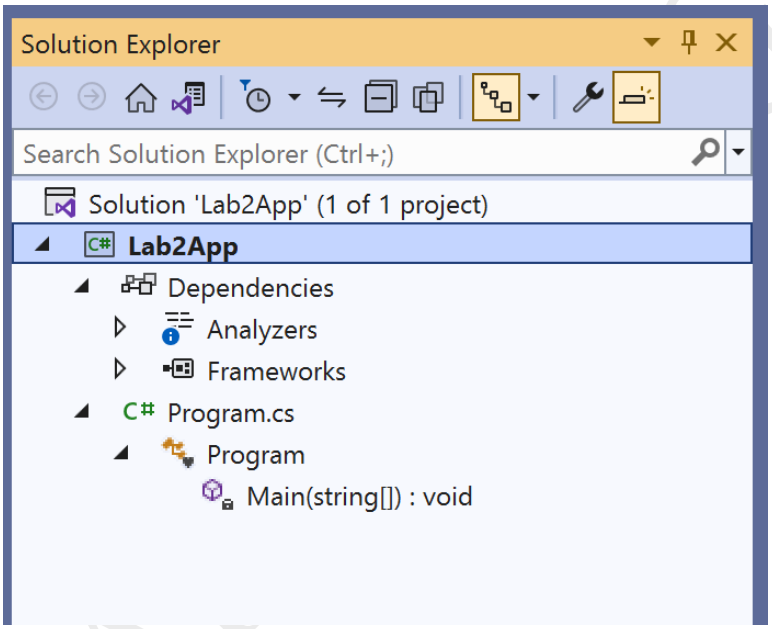
PASUL 3:



PASUL 4:

În fișierul **Program.cs** vom scrie cel mai simplu program în C#. Se va afișa în consolă mesajul “Hello World!”





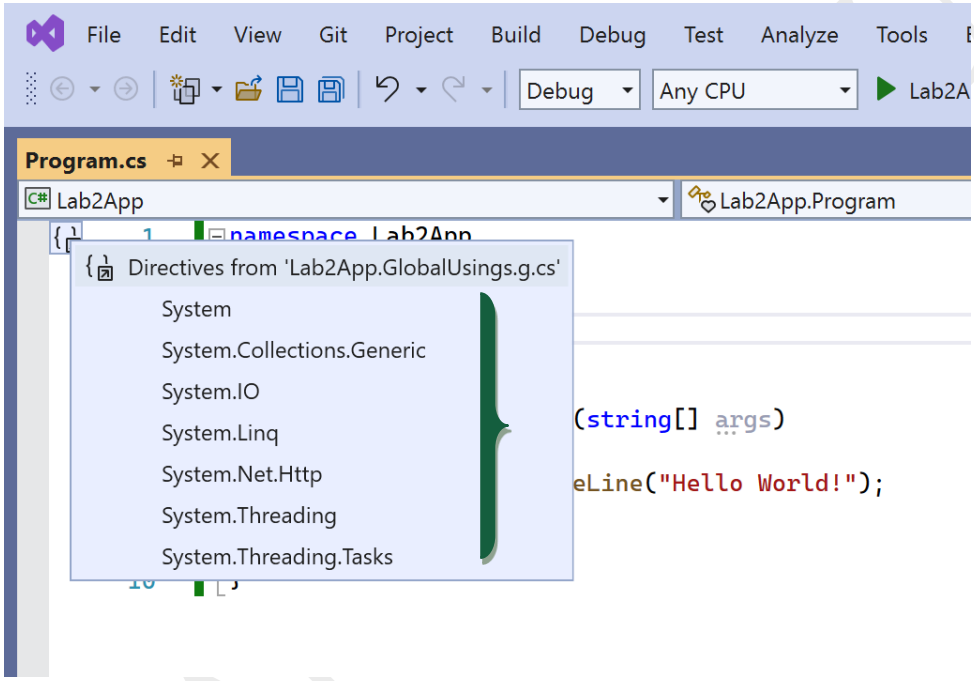
În momentul creării proiectului, acesta a fost numit **Lab2App**. Denumirea proiectului devine automat **namespace**.

Un namespace – este o colecție de clase, **funcții**, **interfețe**, **structuri**, etc. Namespace-ul este utilizat pentru a organiza intern codul și a preveni conflictele de nume, fiind o modalitate de a grupa logic elementele. În același timp, oferă și un mecanism de control asupra vizibilității și utilizării acestora în diferite părți ale programului.

Namespace-ul Lab2App conține **clasa Program**. Atunci când scriem cod într-un namespace, avem acces la toate elementele definite în el.

Dacă se dorește utilizarea unei componente dintr-un alt namespace, atunci acel namespace trebuie importat.

Pentru **import** se utilizează **using**. Visual Studio are incluse câteva astfel de directive implicite.

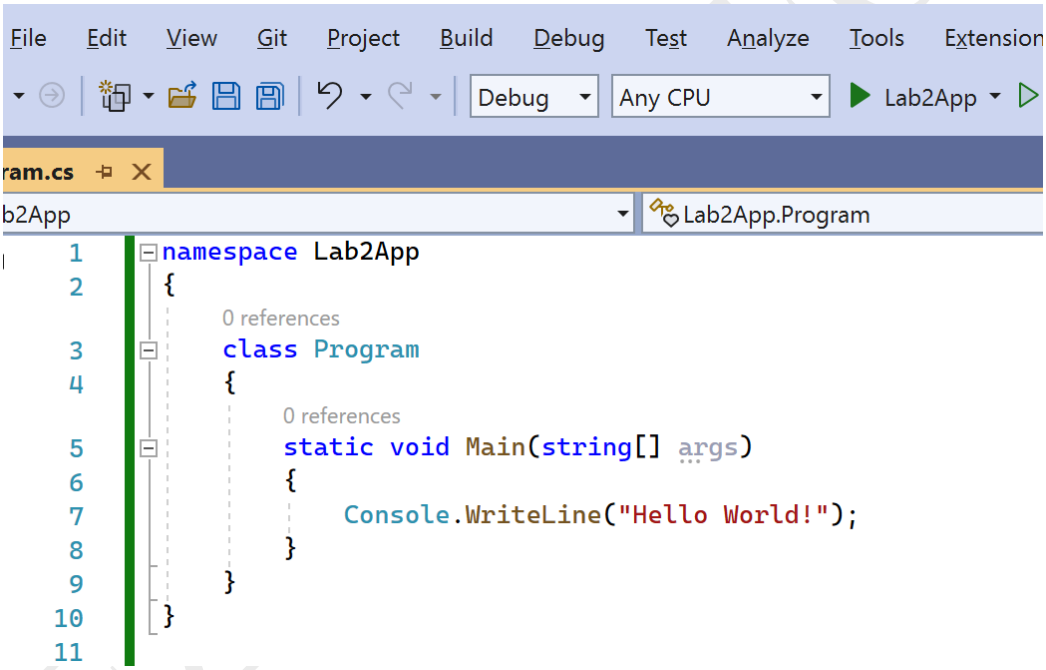


Directiva System – oferă acces la toate clasele de bază. În cazul de față, System oferă acces la **clasa Console**, care la rândul ei conține metoda **WriteLine**.

Acest namespace include (sunt prezentate cele mai utilizate):

- clase pentru **tipurile de date de bază**, precum: int, string, bool, float;
- clase pentru **manipularea excepțiilor** (Exception);
- **conversii** pentru tipurile de date;
- clase pentru **lucrul cu fișiere** (System.IO);
- **colecții** (System.Collections);

- metode pentru **manipularea** și **interogarea colecțiilor de date** (System.Linq) folosind **LINQ** (Language Integrated Query). LINQ este o componentă esențială a limbajului C# care permite lucrul cu date într-o manieră declarativă, similară cu interogarile SQL, dar aplicabilă și în cazul colecțiilor de date (liste, array-uri);
- **System.Net.Http** - namespace care oferă funcționalități pentru a lucra cu protocolul **HTTP** (Hypertext Transfer Protocol), permițând dezvoltatorilor să trimită și să primească date prin intermediul internetului. Acest namespace este fundamental atunci când se lucrează cu apeluri HTTP, cum ar fi solicitările GET, POST, PUT, DELETE.



Clasa Program – definește metoda **Main**. Comparativ cu Java, unde clasa care conține metoda Main trebuie să se numească tot Main, în cazul lui C# clasa poate avea orice denumire.

static void Main (string[] args) – este metoda principală a programului, metodă care se apelează prima.

/!\ OBS:

- C# este case sensitive (**VEZI** mai jos – naming conventions)
- Orice declarație sau expresie se incheie cu **;**
- Execuția programelor începe întotdeauna cu metoda Main
- Comentariul pe o singura linie se include folosind //
- Comentariile pe mai multe linii se includ folosind /**/

Variabile și Tipuri de date

O **variabilă** – este un nume pe care îl ia o zonă de memorie. Ulterior acel spațiu de memorie poate fi manipulat prin intermediul denumirii variabilei.

Fiecare variabilă are un anumit tip de date, care determină dimensiunea zonei de memorie.

Tipuri de date în C#

Cele mai utilizate tipuri de date sunt:

- **int** – numere întregi fără virgulă
- **double** – numere întregi cu virgulă
- **char** – stochează câte un caracter
- **string** – stochează text
- **bool** – stochează valori true sau false
- **object** – tipul obiect este clasa de bază pentru toate tipurile de date în C#. Tipurilor obiecte li se pot atribui valori de orice tip.

```
// TIPURILE DE DATE

  int nr = 100;
  string str = "Acesta este un text";
  double d = 12.35;
  char c = 'a';
  bool b = true;
  object obj = 100;

  Console.WriteLine("Numarul este " + nr);
  Console.WriteLine("Stringul este: " + str);
  Console.WriteLine("Numarul in virgula mobila este: " + d);
  Console.WriteLine("Caracterul este: " + c);
  Console.WriteLine("Valoarea de adevar este: " + b);
  Console.WriteLine("Obiectul este: " + obj);
```

Conversii de tip

Conversiile de tip se împart în două categorii:

- **implicite** – compilatorul C# convertește automat un tip de date în alt tip de date. În general tipuri de date care ocupa o zonă mai mică de memorie, cum este tipul int, sunt convertite automat în tipuri de date care ocupă o zonă mai mare de memorie.

```
// CONVERSII IMPLICITE

int nrInt = 10;

// Metoda GetType() preia tipul de date
Type tipNrInt = nrInt.GetType();

// Conversie implicita
double nrDouble = nrInt;

// Se preia tipul
Type tipNrDouble = nrDouble.GetType();

// Afisare valori inainte de conversie
Console.WriteLine("nrInt value: " + nrInt);
Console.WriteLine("nrInt Type: " + tipNrInt);

// Afisare valori dupa conversia implicita
Console.WriteLine("nrDouble value: " + nrDouble);
Console.WriteLine("nrDouble Type: " + tipNrDouble);
```

Compilatorul a convertit implicit tipul int în double, fără pierdere de informație.

- **explicite** – în cazul în care se dorește conversia unui tip care ocupă o zonă mai mare de memorie, într-un tip care are o dimensiune mai mică a memoriei

```
// CONVERSII EXPLICITE

double nDouble = 25.123;

// Conversie explicita
int nInt = (int)nDouble;

// Afisarea valorii inainte de conversie
Console.WriteLine("Valoarea inainte de conversie a fost: "
+ nDouble);

// Afisarea valorii dupa conversie
Console.WriteLine("Valoarea dupa conversie este: " +
nInt);
```

- Se poate utiliza și conversia folosind **Parse()**. Aceasta se utilizează cu precădere în cazurile în care **se convertesc tipuri de date care nu sunt compatibile**. De exemplu, int și string

  Sintaxa: int.Parse(parametru);

```
// CONVERSIE UTILIZAND PARSE()

string st = "100";

// tipul de date
Type tip1 = st.GetType();

// Se converteste tipul string in int
int x = int.Parse(st);
Type tip2 = x.GetType();


Console.WriteLine("Valoarea initiala a fost: " + st);
Console.WriteLine("A avut tipul: " + tip1);

Console.WriteLine("Noua valoare dupa conversie este: " + x);
Console.WriteLine("Valoarea dupa conversie are tipul: " + tip2);
```

- Conversii folosind **clasa Convert** – clasa pune la dispoziție numeroase metode pentru a converti orice tip de date într-un alt tip de date -> **ToBoolean()**, **ToChar()**, **ToDouble()**, **ToInt16()**, **ToString()**;

```
// CONVERSII FOLOSIND CLASA CONVERT

 int num = 25;
 Console.WriteLine("Valoare de tip int: " + num);

 // Se converteste valoarea int in stringul "25"
 string strConvert = Convert.ToString(num);
 Console.WriteLine("Valoarea dupa conversie " +
strConvert);
 Console.WriteLine("Tipul dupa conversie: " +
strConvert.GetType());

 // Conversie in Double
 Double doubleConvert = Convert.ToDouble(num);
 Console.WriteLine("Valoarea dupa conversie " +
doubleConvert);
                Console.WriteLine("Tipul dupa conversie: " +
 doubleConvert.GetType());
```

Când utilizăm **ToInt16()**?

- Numărul 16 face referire la faptul că numărul rezultat este reprezentat pe **16 biți**, adică poate stoca valori întregi într-un interval limitat specific → de la **-32.768 până la +32.767**

- Folosirea ToInt16() în loc de **ToInt32()** (care convertește într-un int pe 32 de biți) sau **ToInt64()** (pentru long pe 64 de biți) depinde de dimensiunea și natura valorii utilizate. Dacă valoarea o să fie în intervalul unui Int16, atunci aceasta este o alegere mai eficientă din punct de vedere al memoriei.

Nullable

Nullable reprezintă în C# mai multe tipuri de date. Aceste tipuri de date pot lua valori dintr-un interval de valori sau pot fi null.

Sintaxa pentru declararea unui tip de date nullable este următoarea:

<tipul_de_date>? <nume_variabila> = null;

Exemplu:

```
// NULLABLE
  int? num1 = null;
  int? num2 = 45;

  Console.WriteLine("Valorile sunt: {0}, {1}", num1, num2);
```

Instrucțiuni de control

```
// INSTRUCTIUNI DE CONTROL


            // if
            if (Conditie)
            {
                // se executa daca conditia este true
            }



            // if else
            if (Conditie)
            {
                // se executa daca conditia este true
            }
            else
            {
                // se executa daca conditia este false
            }


            // if else --- else
            if (Conditie1)
            {
                // se executa daca conditia1 este true
            }
            else if (Conditie2)
            {
                // se executa daca conditia2 este true
            }
            else if (Conditie3)
            {
                // se executa daca conditia3 este true
            }
            else
            {
                // se executa daca nicio conditie din cele anterioare
nu se indeplineste
            }
```

```
            // Se poate utiliza si -> short-hand if...else
            // se numeste si operator ternar (ternary operator)
            // deoarece este format din 3 operanzi

            // var = (condition) ? expressionTrue : expressionFalse;

             int nr = 10;
             string result = (nr % 2 == 0) ? "Numarul este par" :
"Numarul este impar";
             Console.WriteLine(result);



             // if imbricat (nested)
             if (Conditie1)
             {
                 // se executa daca conditia1 este true
                 if (Conditie2)
                 {
                     // se executa daca conditia2 este true
                 }
             }



             // while loop
             while (Conditie)
             {
                 statement(s);
             }



             // for loop
             for (init; conditie; increment)
             {
                 statement(s);
             }



             // do...while loop
             do
             {
                 statement(s);
              } while(Conditie);
```

Array-uri

Un **array** este utilizat pentru a stoca o **colecție de date de același tip**. Fiecare element din array este accesat prin indexul său.

Sintaxa de declarare a unui array:

datatype[] numeArray;

```
// ARRAY
        int[] n = new int[10];

        for (int i = 0; i < 10; i++)
        {
            n[i] = i + 1;
        }

        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine("Element[{0}] = {1}", i, n[i]);
        }
```

Clase și Obiecte

Clasele și **obiectele** stau la baza Programării Orientate pe Obiecte.

O **clasă** reprezintă o structură de date care conține **proprietăți** și **metode** care se vor aplica asupra datelor, comportându-se ca un tip de date. În ASP.NET, în momentul creării bazei de date, clasele vor reprezenta tabelele din baza de date, fiecare clasă având proprietăți specifice.

Un **obiect** reprezintă o **instanță a unei clase**. La crearea unui obiect, are loc o alocare specifică de memorie pentru a stoca datele și atributele asociate acelui obiect.

Exemplu:

```
public class Employee
{
   // Definirea atributelor(proprietatilor)

   public int EmployeeId { get; set; }
   public string FirstName { get; set; }
   public string LastName { get; set; }
   public decimal Salary { get; set; }


   // Constructor pentru initializarea obiectului Employee

   public Employee(int employeeId, string firstName, string
lastName, decimal salary)

   {
       this.EmployeeId = employeeId;
       this.FirstName = firstName;
       LastName = lastName;
       Salary = salary;
   }

   // Metoda pentru afisarea detaliilor angajatului
   public void Index()
   {
       Console.WriteLine("Employee ID: " + EmployeeId);
       Console.WriteLine("First Name: " + FirstName);
       Console.WriteLine("Last Name: " + LastName);
       Console.WriteLine("Salary: " + Salary);
   }
}

class Program
{
   static void Main(string[] args)
   {
       // Crearea unui obiect al clasei Employee

       Employee emp = new Employee(1, "Pop", "Lucian", 2500);
```

```
            // Apelarea metodei Index prin intermediul obiectului emp,
            // obiectul fiind o instanta a clasei Employee
            // Astfel se afiseaza detaliile angajatului

            emp.Index();
        }
    }
```

Constructor

- Trebuie să aibă același nume ca și clasa;
- Poate fi public, protected sau private;
- O clasă poate avea mai mulți constructori, cu un număr diferit de parametri, dar nu poate avea decât un singur constructor fără parametri;
- Nu poate returna nicio valoare, deci nu are un tip pe care să îl returneze;
- Dacă nu există constructor, compilatorul C# o să creeze unul în mod automat;

This

This este referința la instanța curentă a clasei, dar doar în interiorul clasei în care este utilizat. Se utilizează în momentul în care se dorește accesarea membrilor clasei (proprietăți, metode). În exemplul de mai jos sunt corecte ambele implementări.

```
            this.EmployeeId = employeeId;
            this.FirstName = firstName;
            LastName = lastName;
            Salary = salary;
```

Convenții de nume - Naming conventions

- Se utilizează **PascalCase** pentru:
  - → Clase
  - → Constructor
  - → Metode

- Se utilizează **camelCase** pentru:
  - → Variabile
  - → Argumentele metodelor