### Modulul 3. Clasele în C#

#### Sintaxa declarării claselor în C#

Clasele sunt partea centrală a C#, majoritatea variabilelor și funcțiilor sunt declarate și utilizate în înteriorul claselor, dar când vorbim de declararea unei clase în cod trebuie să înțelegem că aceasta reprezintă doar o referință.

De exemplu codul

ClasaMea clasa;

Va fi echivalentă cu null atâta timp cât programatorul nu va defini la ce face referință variabila clasa.

Din acest motiv de fiecare dată când vrem să utilizăm o clasă trebuie să o definim manual ce funcție are aceasta și cum va fi creată.

*Sintaxa declarării unei clase în C#*

**//Nivelul de acces - clasă - denumirea clasei - clasa sau interfața de la care moștenește**

public class ClasaMea: ClasaBase

{

//Variabile, metode etc.

}

#### Nivelul de accesibilitate al claselor în C#

|  |  |
| --- | --- |
| **Nivelul de accesibilitate** | **Explicație** |
| [public](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/public) | Acces liber. |
| [protected](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/protected) | Acces în interiorul clasei sau în claselor derivate. |
| [internal](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/internal) | Acces în interiorul clasei și în clasele din același assembly. |
| [protected internal](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/protected-internal) | Acces în interiorul clasei, în cadrul claselor derivate și în cadrul claselor din același assembly. |
| [private](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/private) | Acces doar în interiorul clasei. |
| [private protected](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/private-protected) | Acces în interiorul clasei, și în clasele derivate DOAR dacă fac parte din același assembly.  Disponibil de la lansarea C# 7.2. |
| [sealed](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/abstract-and-sealed-classes-and-class-members) | Clasa cu nivelul dat nu permite altor clase să moștenească de la ea (îmbunătățește un pic performanța) |
| [abstract](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/abstract-and-sealed-classes-and-class-members) | Nu poate fi creată prin constructori sau folosită ca clasă statică. O clasă abstractă reprezintă un fundament pentru alte clase care vor moșteni de la ea. |

#### Variabilele claselor

Câmpurile unei clase reprezintă toate variabilele care sunt incluse în această clasă. O distincție importantă când discutăm de câmpuri este să înțelegem diferența dintre varabilele statice și nestatice.

*De exemplu:*

public class ClasaMea

{

public int nrOrdine; //Unul și același pentru toate obiectele ce dețin ClasaMea

public string denumire; //Diferit pentru fiecare obiect de tip ClasaMea

}

Pentru optimizarea mai bună a codului se recomandă să declarăm câmpurile în interiorul clasei doar când avem nevoie de ele în mai multe funcții/clase.

În mod opus se recomandă să declarăm variabilele în mod local doar în interiorul funcțiilor.

#### Funcțiile claselor

O funcție în C# este un bloc de cod care este executat la derularea programului. *Aceasta poate începe de la cel mai simplu:*

private static void Main()

{

Console.WriteLine("a");

}

Sau poate include sute de variabile, operații, utilizarea inclusă a altor funcții, referințe spre alte clase exterioare etc.

***Se recomandă ca o funcție să fie numită conform funcției pe care o execută și să conțină doar codul care îndeplinește acest scop.***

1. **Funcții cu parametri**

Este un tip de funcții tipizat care primește ca input(date de intrare) anumite date, sau referințe care le numim parametri.

Funcțiile cu parametri pot simplifica semnificativ lucrul programatorilor din motiv că ele automatizează unele comenzi care se folosi în multiple locuri, sau mai ales în mai multe clase.

1. **Declarația return**

Declarația return are 2 funcționalități în C#:

1. Termină execuția unei funcții și returnează controlul funcției care a apelat-o
2. Poate returna o anumită variabilă.

***Dacă funcția este de tip void, declarația return poate fi omisă, dacă nu funcția return este obligatorie.***

1. **Supraîncărcarea funcțiilor**

Ca și în alte limbaje orientate pe obiect în C# supraîncărcarea se folosește pentru a implementa polimorfismul, adică atunci când avem nevoie ca o funcție să posede mai multe forme.

public class Program

{

public int Add(int a, int b)

{

int sum = a + b;

return sum;

}

public int Add(int a, int b, int c)

{

int sum = a + b + c;

return sum;

}

public static void Main(String[] args)

{

//Creeam obiectul

Program ob = new Program();

int sum1 = ob.Add(1, 2);

Console.WriteLine("Suma a doua numere intregi este:{0}", sum1);

int sum2 = ob.Add(1, 2, 3);

Console.WriteLine("Suma a trei numere intregi este:{0}", sum2);

}

}

1. **Sintaxa scurtă a funcțiilor de un rând și operatorul =>**

Operatorul => în C# poate fi utilizat pentru scrie pe scurt o funcție care returnează o variabilă din interiorul clasei.

public class Oras

{

private string numeleOrasului = “Chisinau”;

public string nume=> numeleOrasului ;

}

#### Constructori

1. Definiția unui constructor

Oricând este creată o clasă sau o structură în C# este utilizat un constructor. O clasă poate avea mai mulți constructori predefiniți de programator pentru a-i oferi o gamă mai mare de posibilități.

Observații și reguli:

1. ***Dacă nu creați un constructor manual atunci C# va crea și utiliza un constructor implicit fără parametri.***
2. ***Un constructor totdeauna are același nume ca și clasa.***
3. ***Un constructor poate să fie fără parametri sau cu parametri multipli***
4. Constructor cu parametri
5. Constructor supraîncărcat

#### Cuvântul cheie this

Cuvântul cheie this în C# se folosește atunci când avem nevoie să facem referință directă la obiectul curent, ceea ce devine extrem de important în cazul în care am folosit un constructor pentru a crea mai multe obiecte cu aceeași clasă.

public class Angajat

{

public string alias;

public string name;

public Employee(string name, string alias)

{

this.name = name;

this.alias = alias;

}

}

#### Utilizarea cuvintelor cheie ref

1. Utilizarea cuvântului cheie ref

Cuvântul cheie ref în C# se folosește pentru a transmite referințe către o anumită metodă, ceea ce semnifică că parametrul trimis către funcție nu va fi modificat doar în funcția locală dar și la sursa acestuia.

***Exemplu:***

class VarReferinta

{

static void Main(string[] args)

{

int a = 10;

int b = 12;

Console.WriteLine("Valoarea initiala a este {0}", a);

Console.WriteLine("Valoarea initiala b este {0}", b + "\n");

//Apelam functia exterioara adauga cu a, care este transmis ca parametru simplu

adauga(a);

Console.WriteLine("Valoarea dupa adaugare {0}", a);

//Apelam functia exterioara scade cu b, care de data aceasta este transmis ca referinta si modificat in functia main de asemenea

scade(ref b);

Console.WriteLine("Value dupa scadere, cu ref: {0}", b);

}

public static void adauga(int a)

{

a += 10;

}

public static void scade(ref int b)

{

b -= 5;

}

}

**Întrebări rămase din timpul lecției și exemple adiționale.**

**1.** **Când folosim cuvântul cheie this și care e utilitatea**

**a)** **Ca parametru al unei funcții(cea mai utilă utilizare)**

using System;

namespace ThisExemplu

{

class Employee

{

public string FirstName;

public string LastName;

public int Salary;

public Employee(string \_firstName, string \_lastName, int \_salary)

{

FirstName = \_firstName;

LastName = \_lastName;

Salary = \_salary;

}

private static void Main()

{

Employee employee = new Employee("John", "Carmack", 15000);

employee.SaveData();

}

public void SaveData()

{

DataStorage.Store(this);

}

}

class DataStorage

{

public static void Store(Employee employee)

{

Console.WriteLine("Employee {0} {1} has a monthly salary of {2}$.", employee.FirstName, employee.LastName, employee.Salary);

}

}

}

**b)** **Când avem nevoie să returnăm o referință**

using System;

namespace ThisExemplu

{

class Program

{

public string mesaj = "Ai obtinut o referinta cu ajutorul this :)";

public Program Referinta()

{

return this;

}

}

class Program2

{

public static Program programBaza;

private static void Main()

{

Program program = new Program();

programBaza = program.Referinta();

Console.WriteLine(programBaza.mesaj);

}

}

}

**c)** **Când vrem să evităm erorile legate de numele ambiguu al variabilelor(putem evita aceste erori și prin metode mai simple dar această variantă e preferată de unii programatori)**

using System;

public class Employee

{

private string alias;

private string name;

public Employee(string name, string alias)

{

this.name = name;

this.alias = alias;

}

private static void Main()

{

Employee newEmployee = new Employee("Ron Harper", "RH");

Console.WriteLine("The new employee's name is {0} and his alias is {1}.", newEmployee.name, newEmployee.alias);

}

}

**2.** **Cum creăm și folosim un singleton și când îl folosim**

**a)** **Vom folosi un parametru singleton atunci când vrem să ne asigurăm că va exista doar o singură referință la o anumită clasă.**

**Exemplu:** În cazul de mai jos pentru clasa exemplu am încercat să creez două referințe și să incrementez variabila publică număr cu diferite valori, dar dacă veți da la execuție programul veți vedea că obținem aceeași valoare pentru ambele referințe. Vom discuta de ce în timpul lecției.

using System;

namespace Singleton

{

public class Exemplu

{

public static Exemplu instance = null;

private Exemplu() { }

public static Exemplu GetInstance

{

get

{

if (instance == null)

{

instance = new Exemplu();

}

return instance;

}

}

//Variabila locala

public int numar = 0;

}

public class ImplementareExemplu

{

private Exemplu referinta = Exemplu.GetInstance;

private Exemplu referinta2 = Exemplu.GetInstance;

private static void Main()

{

ImplementareExemplu impl = new ImplementareExemplu();

impl.IncrementNumar(1);

impl.IncrementNumar(2);

impl.IncrementNumar(3);

}

private void IncrementNumar(int parametru)

{

referinta.numar += parametru;

referinta2.numar += parametru \* 2;

Console.WriteLine("Referinta 1:" + referinta.numar);

Console.WriteLine("Referinta 2:" + referinta2.numar);

}

}

}