### Modulul 5. Namespace-urile și proprietațile lor în C#

#### Ce este un namespace?

Namespace este un cuvânt cheie în C# utilizat pentru a declara un “container” care poate conține:

* Alte namespace-uri
* Clase
* Interfețe
* Structuri
* Delegate(pointeri spre funcții în C++)

Pentru a scrie un program în C# nu este necesar de a folosi namespace-uri dar scopul lor principal este de a împărți codul vostru în mai multe părți structurate.

Cea mai simplă analogie pentru a demonstra utilitatea namespace-urilor este un fișier. Când aveți un proiect mare, care conține multe documente, file-uri și alte elemente este posibil de a le include pe toate într-un singur fișier, sau să le lăsați chiar pe desktop, dar ar fi mult mai ușor dacă toate aceste elemente erau grupate după relevanță și plasate în fișiere separate, în C# namespace-uri servesc un rol similar cu cel al fișierelor menționate mai sus.

***Este important de menționat că utilitatea namespace-urilor crește exponențial odată cu dimensiunea proiectului vostru.***

*Exemplu 1:*

namespace Lectia5\_Namespace

{

using System;

namespace Namespace\_Inclus

{

class Clasa1

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

}

}

}

interface InterfataExemplu

{

}

struct StructuraExemplu

{

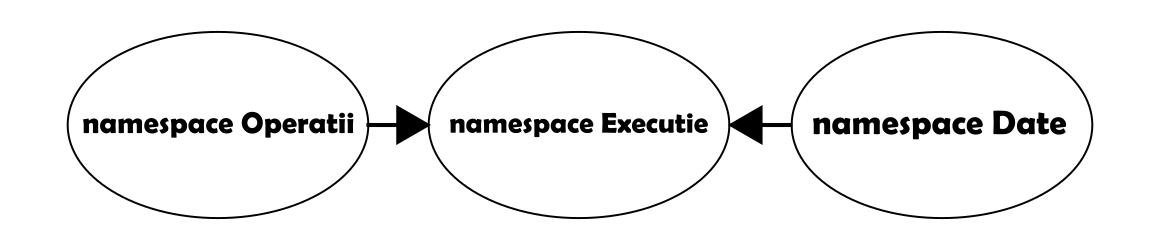
}

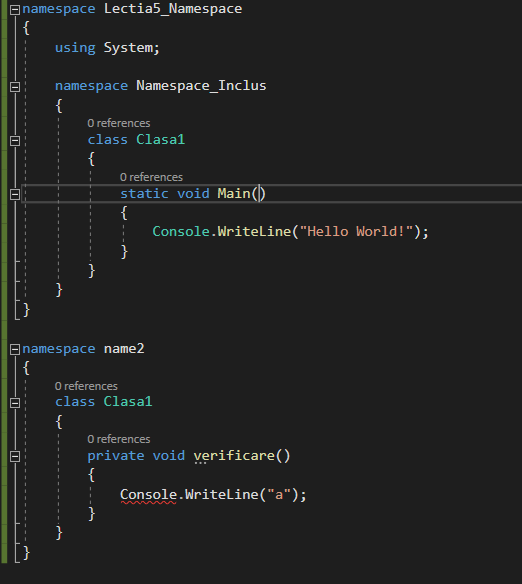
public delegate int PerformCalculation(int x, int y);

}

#### Scopurile și importanța utilizării namespace-urilor

1. Pentru a organiza clasele
2. Pentru a evita conflictele de nume între clase
3. Pentru a gestiona la ce librării au acces clasele din namespace

**



#### Declararea namespace-urilor

Un namespace poate fi declarat chiar în primul rând al programului, sau după indicarea librăriilor pe care le veți utiliza.

|  |  |
| --- | --- |
|  |  |

Vom utiliza cuvântul cheie *namespace* după care vom indica numele selectat. Se recomandă ca namespace-urile voastre să fie denumite cu literă mare și conform funcțiilor care le execută

*De exemplu:*

* **namespace SalvareaDatelor** - în care vom include clasele, structurile și interfețele pentru salvarea datelor
* **namespace AfisareaDatelor** - ce va conține clasele.. pentru afișarea datelor șamd.

#### Cuvântul cheie using

De fiecare dată când faceți referință la un namespace puteți indica numele acestuia la fiecare referință. Dar cuvântul cheie using face acest lucru cu mult mai simplu.

Esența cuvântului cheie using este de a indica care librării și namespace-uri sunt disponibile.

*De exemplu:*

* **using System** - programatorul poate scrie manual referințele la librăria de sistem, ca în cazul System.Console.WriteLine(“XYZ”); dar în cazul în care avem nevoie să facem această referință de mai multe ori este rezonabil să indicăm *using System* și să reducem cantitatea de cod și timp necesară pentru a crea o referință.

#### Namespace-uri incluse

În C# un namespace poate include alt namespace în interior. În acest caz namespace-ul din interior va fi numit namespace inclus.

Acest tip de namespace-uri vor fi folosite atunci când programul va avea nevoie de mai multe nivele de structură.

Atunci când folosim un namespace inclus trebuie să includem numele namespace-ului de bază după care punem . și numele namespace-ului inclus.

*Exemplu 1:*

**using System;**

**using System.Collections;**

*Exemplu 2:*

namespace Operatii

{

namespace OperatiiAritmetice

{

class Class1

{

public static int Adunare(int a, int b)

{

int sum = a + b;

return sum;

}

}

}

}

namespace Baza

{

class Program

{

static void Main()

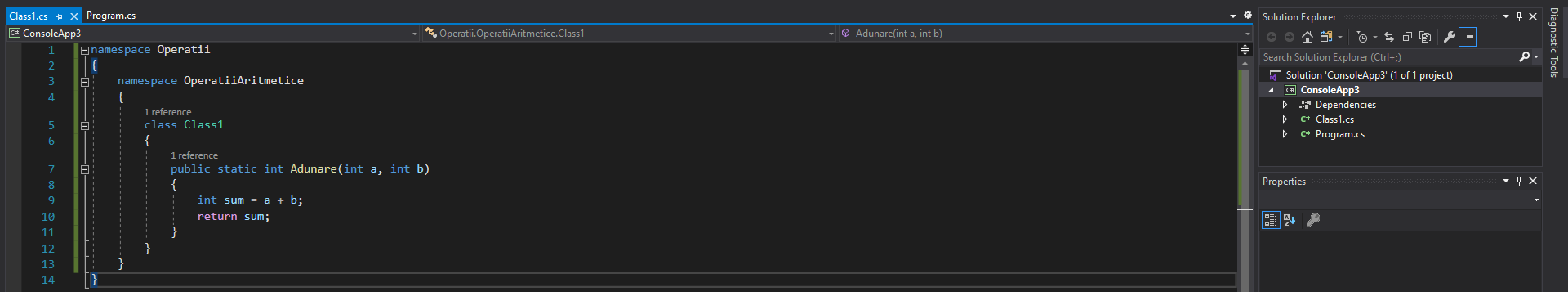
{

Console.WriteLine(Operatii.OperatiiAritmetice.Class1.Adunare(1,3).ToString());

}

}

}

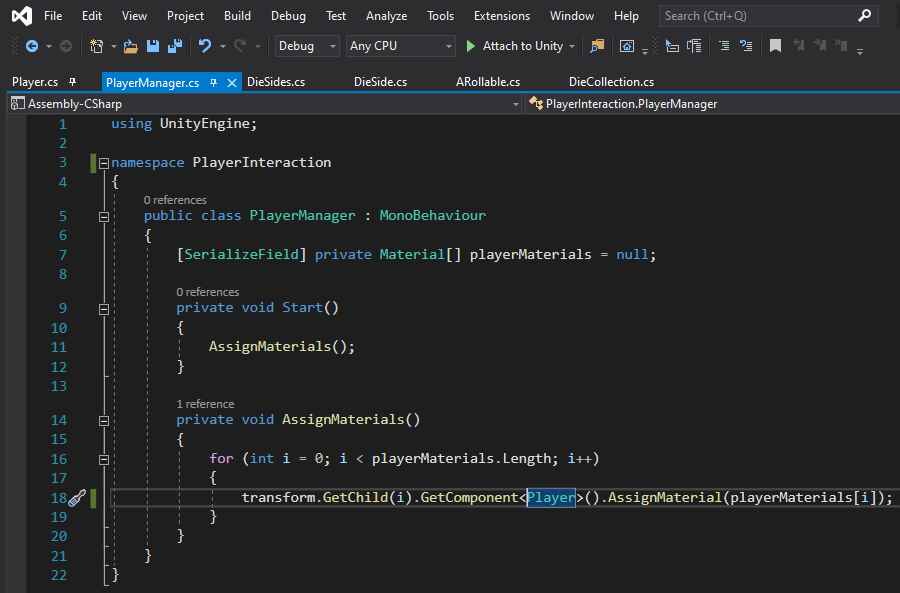


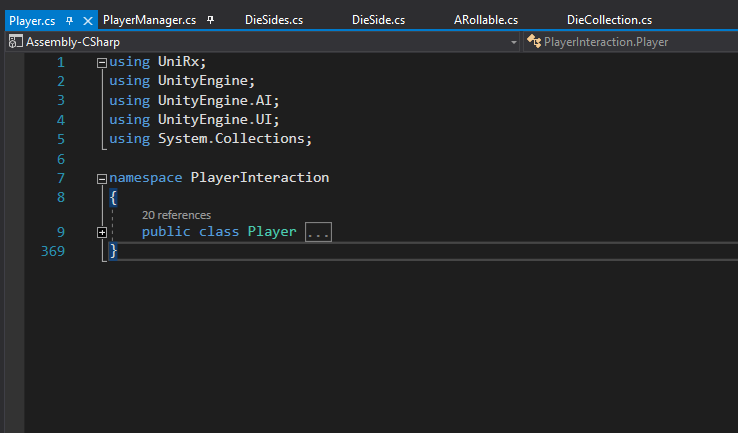
#### Împărțirea namespace-urilor în mai multe părți

În C# un singur namespace poate fi divizat în mai multe părți. La prima vedere acest lucru poate părea inutil dar, atunci când aveți un proiect mare, la care lucrează o echipă mare, codul nu trebuie stocat în același fișier, altfel vom obține un fișier de zeci sau sute de mii de rânduri.

Din acest motiv este recomandat să divizați namespace-urile voastre în mai multe fișiere.

*Exemplu:*

**

**

Mai sus vedem exemplul unui program C# care are o multitudine de clase. Două din ele: Player și PlayerManager sunt legate logic și interacționează între ele într-o oarecare măsură.

Dar în același timp Player.cs gestionează comportamentul unui jucător individual, iar PlayerManager ține cont de toți jucătorii simultan, acesta este primul motiv de a le separa în diferite fișiere.

Al doilea este că clasa Player este destul de voluminoască(369 rânduri) și dacă am include și clasa PlayerManager în ea am obține un document mare, cu logică ce ține de lucruri și scopuri diferite, ceea ce trebuie evitat cu orice preț.

Din motivele exprimate am decis să păstrez clasele în aceeași librărie, dar să le separ ca fișiere.

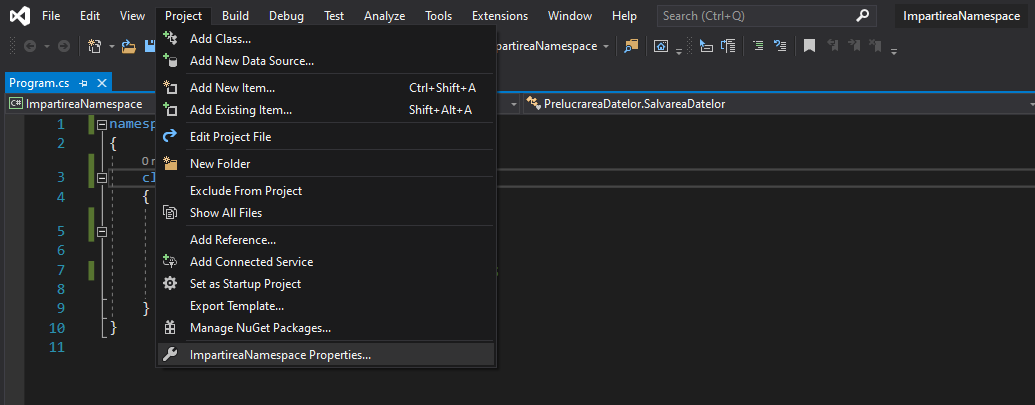
#### Namespace-uri default

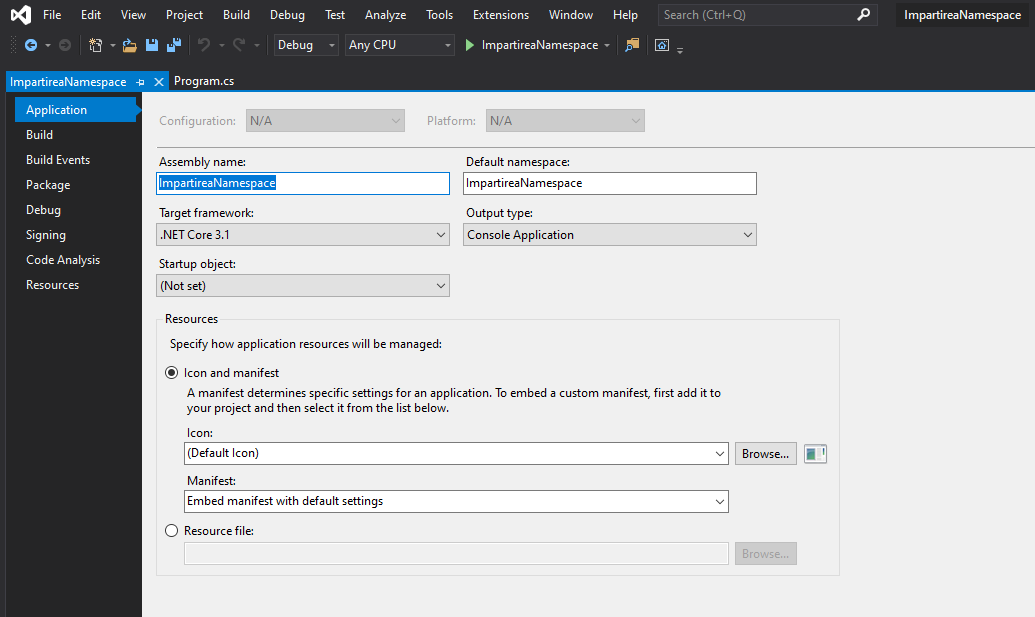
După cum ați observat de fiecare dată când creați un proiect nou în Visual Studio este adăugat automat un namespace care are nume egal cu numele proiectului vostru. Acest namespace se numește default sau global.

Puteți modifica oricând acest namespace fără repercursiuni, cu excepție dacă deja ați făcut referință la el deja, în acest caz va trebui să modificați numele tuturor referințelor.

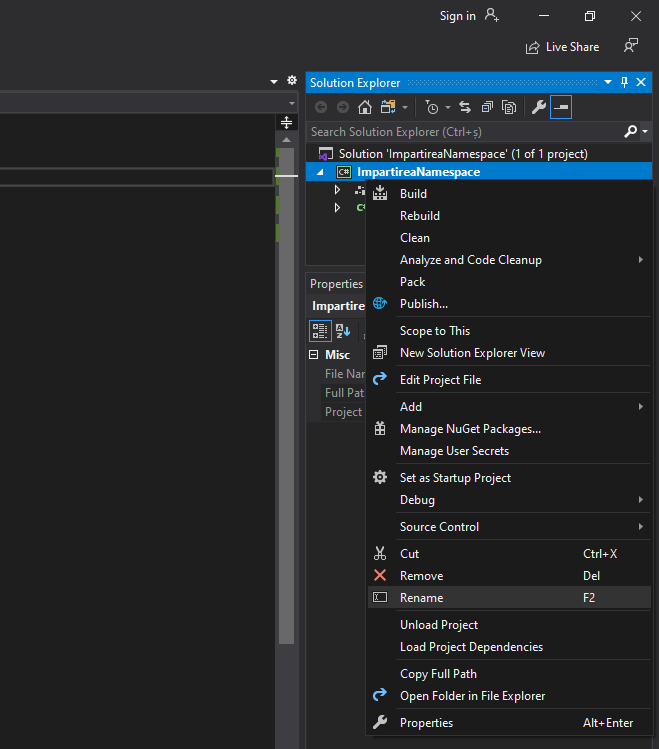
Există 2 metode a schimba namespace-ul default al proiectului dvs.

1. Din proprietățile proiectului **Project-> Properties**. Apoi câmpul **Default Namespace**





1. Sau din fereastra **Solution Explorer din dreapta**, apasând click dreapta apoi **Rename**.



#### A doua formă using și cum o folosim

În C# cuvântul cheie using are 3 utilizări. Prima deja o cunoașteți și este legată de includerea librăriilor care vor fi folosite în fișier/namespace.

A doua utilizarea sau formă a using este folosită pentru a abrevia/presupune utilizarea unei libării, dar sub un nume stabilit de utilizator.

*De exemplu:*

using WinForms = System.Windows.Forms;

using Sys = System;

using Coll = System.Collections;

Această formă este foarte similară cu cuvântul cheie #define în C++ și îndeplinește aceeași funcție. Adițional putem folosi această formă pentru a face referință nu doar la librării, dar și direct la clasele din interiorul acestora, ceea ce face accesul și mai ușor.

*De exemplu:*

using System;

using Date;

using op = Operatii.OperatiiAritmetice;

namespace Executie

{

class Executie

{

static void Main()

{

Info info = new Info(1, 2);

int rezultat = op.Adunare(info.a, info.b);

Console.WriteLine(rezultat);

}

}

}

#### Utilizarea using pentru includerea membrilor statici

A treia formă în care putem folosi cuvântul cheie static este pentru includerea membrilor statici. În acest caz vom include ca de obicei **using** dar după el nu numele librăriei/clasei dar deja cuvântul cheie static. **using static**

Și în final vom include numele clasei pe care vrem să o accesăm în mod static.

**Sintaxa: using static System.Console**

Această utilizare simplifică foarte mult scrierea codului pentru că permite omiterea numelui clasei utilizate, dar în același timp crește riscul de apariție a unor conflicte legate de numele metodelor/claselor folosite dacă apar duplicări.

#### 

#### BONUS: Utilizarea timpului, datei și a timerelor în C#

C# conține multiple librării și metode care vă permit să faceți operații și să comparați timpul, data, să creați timere pentru așteptare etc.

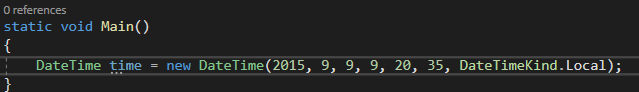
1. **DateTime -** este o structură implementată în C# care permite să creați date noi în care includeți anul, luna, ziua, ora etc., sau extrageți informații despre date deja create, inclusiv despre data curentă.

Valoarea minimă a unui DateTime în C# este 1 Ianuarie, 0001, 00:00:00. Valoarea maximă este 31 Decembrie, 9999, 11:59:59.

Pentru a declara un DateTime nou este suficient să inițializăm un constructor simplu

DateTime dt1 = new DateTime();

Dar putem alege să introducem în acest constructor mai mulți parametri.



***Parametri statici DateTime:***

DateTime currentDateTime = DateTime.Now; //returnează timpul curent exact(zona 0)

DateTime todaysDate = DateTime.Today; //Returnează ziua de azi(ora 12 noaptea)

DateTime currentDateTimeUTC = DateTime.UtcNow;//returnează timpul curent(zona+2)

DateTime maxDateTimeValue = DateTime.MaxValue;//returnează valoarea maximă

DateTime minDateTimeValue = DateTime.MinValue;//returnează valoarea minimă

***Operații cu variabile DateTime:***

1. Adunare, scădere - vor returna o variabilă de tip TimeSpan(interval)
2. Comparații logice - vor returna true/false

***Exemplu:***

using System;

using static System.Console;

namespace TimeTest

{

class Program

{

static void Main()

{

DateTime time = new DateTime(2015, 9, 9, 9, 20, 35, DateTimeKind.Local);

DateTime time2 = new DateTime(2016, 9, 9, 9, 20, 35, DateTimeKind.Local);

TimeSpan rezultat = time2 - time;

WriteLine(rezultat.Days);

}

}

}

***Exemplu 2:***

using System;

using static System.Console;

namespace TimeTest

{

class Program

{

static void Main()

{

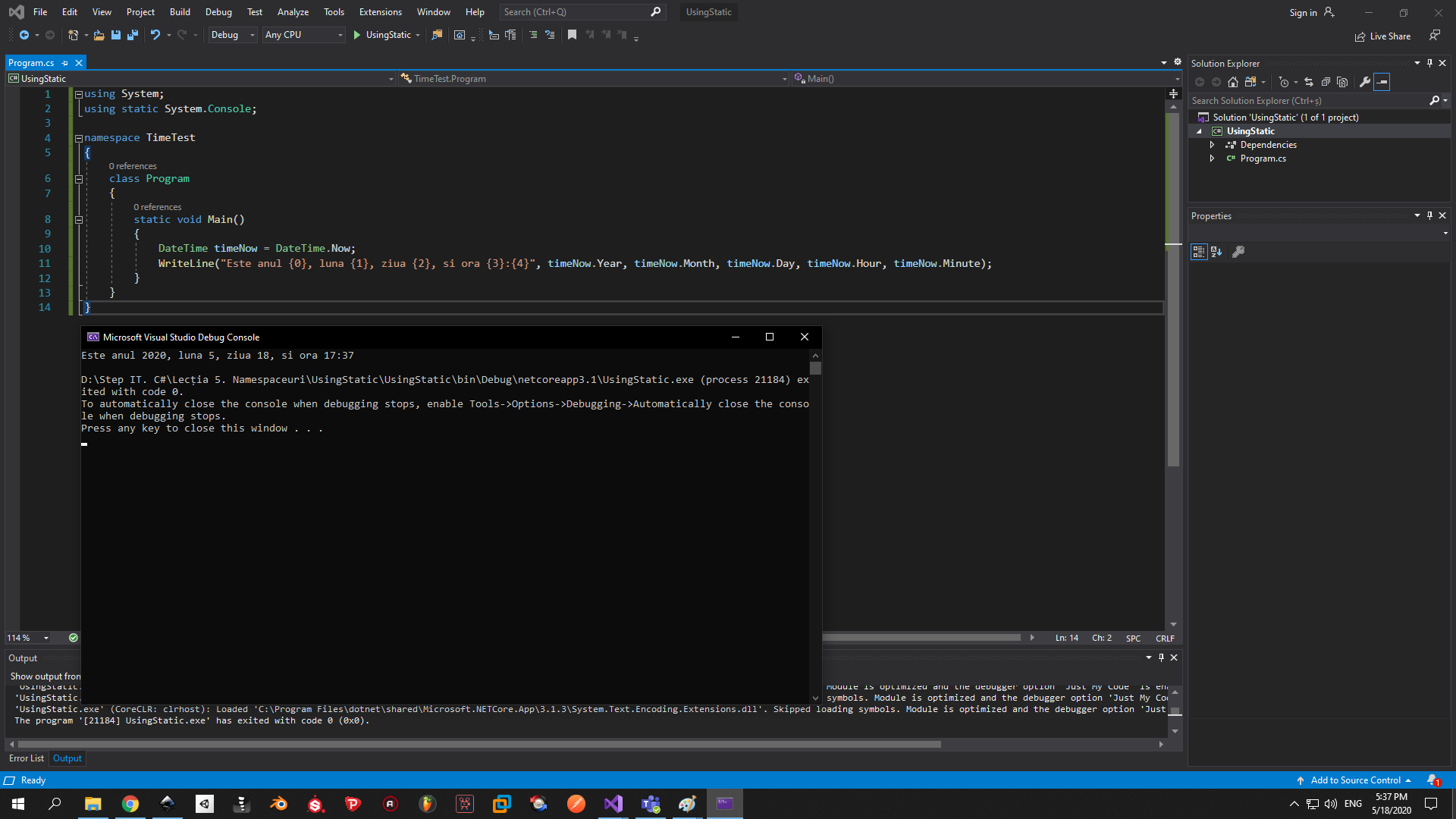
DateTime timeNow = DateTime.Now;

WriteLine("Este anul {0}, luna {1}, ziua {2}, si ora {3}:{4}", timeNow.Year, timeNow.Month, timeNow.Day, timeNow.Hour, timeNow.Minute);

}

}

}



#### Programe propuse pentru elaborare.

**Program 1:**

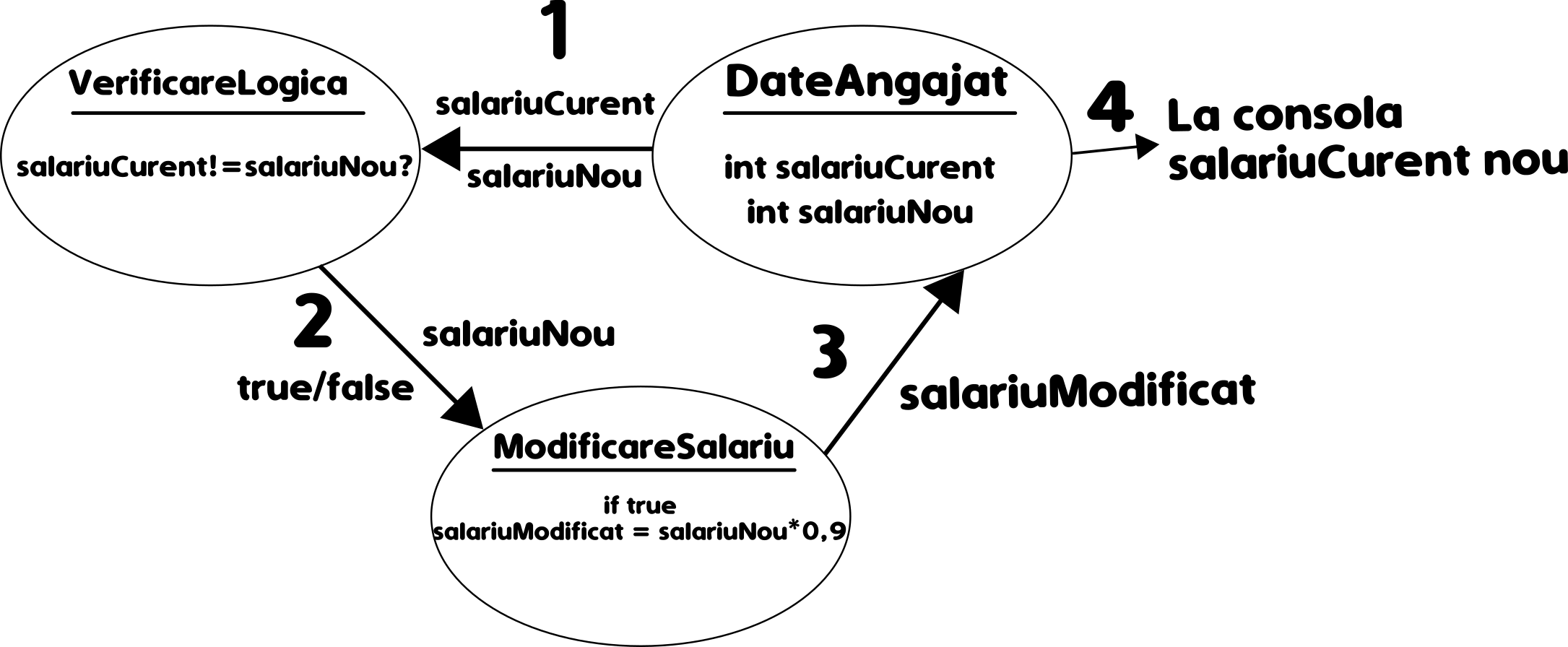
1. Creați un proiect nou în Visual Studio. Creați două clase(în fișiere separate).
2. Introduceți prima clasă într-un namespace cu denumirea **Operații**, și creați metode pentru 4 operații matematice.
3. Introduceți a doua clasă în alt namespace cu denumirea **Execuție**, în interior creați două variabile numerice și executați toate 4 operații din clasa 1.

**Program 2:**

1. Creați un proiect nou în Visual Studio. Creați trei clase(în fișiere separate).
2. Introduceți prima clasă într-un namespace cu denumirea **DateAngajat**, și creați 2 variabile, salariu curent și salariu nou.
3. Introduceți a doua clasă într-un namespace cu denumirea **VerificareLogică.** Creați o metodă care va primi două variabile numerice(salariul curent și cel nou) și v**a returna true dacă salariul nou este diferit de cel curent** și **cât trebuie să fie valoarea nouă a salariului.**
4. Introduceți a treia clasă într-un namespace cu denumirea **ModificareSalariu.** Creați o metodă care primește doi parametri:
5. bool care va indica dacă e nevoie de modificat salariul unui angajat,
6. int de tip simplu care reprezintă salariul nou.

Dacă primim true salariulModificat va fi egal cu 90% din salariulNou primit.

1. Returnați salariul nou clasei din DateAngajat, egalați salariul curent cu numărul primit și printați la consolă rezultatul.



**Program 3:**

1. Creați un proiect nou în Visual Studio. Creați două clase(în fișiere separate).
2. Prima clasă va fi în namespace-ul execuție, iar a doua în namespace-ul calcule.
3. În clasa a doua(responsabilă de calcule) creați o metodă cu doi parametri int și va returna rezultatul împărțirii dintre ele.
4. Din clasa 1(responsabilă de execuție) apelați metoda din clasa 2 și trimiteți ca parametru o cifra int oarecare și 0 ca parametru secundar. Pentru a obține o excepție.
5. În clasa 2 folosiți try pentru a identifica excepția și throw pentru a arunca excepția în clasa 1(responsabilă de execuție).
6. Afișați tipul excepției din clasa 1.

**Program 4:**

1. Creați un proiect nou în Visual Studio. Creați două clase(în fișiere separate).
2. Prima clasă va fi în namespace-ul Afișare, a doua în alt namespace Editare
3. În clasa 1(legată de afișare) declarați un tablou de stringuri.
4. În clasa 2(legată de editare) creați o metodă care primește ca parametru un tablou de string-uri **referință(ref string[])** și dacă găsește în acest tabloul un cuvânt X îl înlocuiește cu Y(la alegerea voastră).
5. Din clasa 1 apelați funcția din clasa 2 și transmiteți ca parametru tabloul local.
6. Afișați tabloul după modificări în clasa 1(legată de afișare).