

C# .NET

Laborator 4

StringBuilder

StringBuilder

- String – imutabile
 - Probleme cu realocarea memoriei si fragmentarea ei
 - Probleme de performanta la operatii frecvente
- Clasa [StringBuilder](#)
 - Lucreaza cu string-uri **mutabile**
 - Optimizata pentru performanta in operatii frecvente
- Folositi StringBuilder
 - Operatii multiple (ex:bucle)
 - Nu se cunoaste de la inceput numarul de modificari
- Nu folositi StringBuilder
 - Operatii foarte putine
 - Optimizari date de compilator
 - Cautari in cadrul string-ului

StringBuilder - operatii

- [StringBuilder Class \(System.Text\) | Microsoft Docs](#)
- Append, AppendFormat, AppendJoin, AppendLine
- Clear
- Insert
- Remove
- Replace
- Length
- Capacity
- **ToString**

Formatarea String-urilor

- compune siruri de caractere pe baza altor informatii / variabile.
 - Similar sirurilor verbatim

```
using System;
```

0 references

```
public class Program
```

```
{
```

0 references

```
public static void Main()
```

```
{
```

```
    int quantity = 10;
```

```
    double unitPrice = 20.5;
```

```
    string productName = "Product A";
```

```
    string unitOfMeasure = "buc";
```

```
    string message = string.Format("Ati cumparat {0} {1} din produsul \"{2}\", pretul unitar fiind {3}", quantity, unitOfMeasure, productName, unitPrice);
```

```
    Console.WriteLine(message);
```

```
}
```

```
}
```

Formatarea String-urilor

- “Variabilele” care vin inlocuite in sablonul specificat ca argument pentru functia format sunt specificate dupa “index” (positional):
 - {0} vine inlocuit cu primul argument de dupa sablon, {1} cu al doilea, {2} cu al treilea, etc...
- Este foarte important sa aveti grija ca “variabilele” care vin inlocuite sa aibe indexul corect (sa fie in acelasi numar, ordinea sa fie corecta).
- In cazul in care specificati in sablon un “index” pentru care nu ati oferit ca parametru si valoarea / variabila, functia string.Format va arunca exceptie
- O anumita “variabila” poate fi repetata de mai multe ori in cadrul unui sablon, sau poate fi omisa.
- Functia string.Format ofera suport pentru diverse [formatari avansate](#):
 - Formatarea valorilor zecimale, a datelor calendaristice, etc
 - Formatarea spatiilor
 - Formatarea alinierilor

Compare, equals,
empty test

Culturi

- Fiecare combinatie de alfabet-Tara reprezinta o cultura
 - [Lista a culturilor](#)
- Cultura default a aplicatiei
 - Data de sistemul de operare
- Afecteaza operatii asupra string-urilor : transformari din litera mare in litera mica, comparatii, parasari de string-uri (DateTime)
-
- Se poate schimba cultura firului de executie astfel:

```
Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-UK");
```


String – compare

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        string str1 = "Bob";
        string str2 = "Alice";

        int compareResult1 = string.Compare(str1, str2);

        if(compareResult1 < 0)
        {
            Console.WriteLine("\"" + str1 + "\" este mai mic decat \"" + str2 + "\"");
        }
        else if (compareResult1 == 0)
        {
            Console.WriteLine("\"" + str1 + "\" este egal cu \"" + str2 + "\"");
        }
        else if (compareResult1 > 0)
        {
            Console.WriteLine("\"" + str1 + "\" este mai mare decat \"" + str2 + "\"");
        }
    }
}
```

String – compare

- Analogie cu regulile de ordonare alfabetica a string-urilor
 - [Detalii aici](#)
- Rezultatul compararii a 2 string-uri poate depinde de:
 - Setarile curente de cultura (limba / regiune)
 - Daca dorim sa tinem cont pur si simplu doar de codurile numerice (ordinal) sau de regulile specifice limbii curente (linguistic)
 - Daca dorim sa tinem cont de casing (litere mari / litere mici)

class Program

{

0 references

static void Main(string[] args)

{

string str1 = "ăăă";

string str2 = "bbb";

Console.WriteLine("Ordinal, case insensitive comparison:");

int compare1 = string.Compare(str1, str2, StringComparison.OrdinalIgnoreCase);

if(compare1 < 0)

{

Console.WriteLine("\"" + str1 + "\" este mai mic decat \"" + str2 + "\"");

}

else if (compare1 == 0)

{

Console.WriteLine("\"" + str1 + "\" este egal cu \"" + str2 + "\"");

}

else if (compare1 > 0)

{

Console.WriteLine("\"" + str1 + "\" este mai mare decat \"" + str2 + "\"");

}

Console.WriteLine("Linguistic, case insensitive comparison:");

int compare2 = string.Compare(str1, str2, StringComparison.CurrentCultureIgnoreCase);

if (compare2 < 0)

{

Console.WriteLine("\"" + str1 + "\" este mai mic decat \"" + str2 + "\"");

}

else if (compare2 == 0)

{

Console.WriteLine("\"" + str1 + "\" este egal cu \"" + str2 + "\"");

}

else if (compare2 > 0)

{

Console.WriteLine("\"" + str1 + "\" este mai mare decat \"" + str2 + "\"");

}

}

}

String – testul egalitatii

- La fel ca pentru comparatie, atunci cand nu specificati alte optiuni, testul de

egalitate este facut cont astfel:

- Se tine cont de casing (case sensitive)
- Compararea tine cont de codurile numerice ale caracterelor (ordinal)
- Pentru specificarea explicita a modului in care se face testul de egalitate, se vor folosi functiile Equals / EqualsTo care permit specificarea parametrului [StringComparisson](#)
- == 😊

```
using System;
using System.Globalization;
using System.Threading;

namespace ConsoleApp2019
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            string str1 = "file";
            string str2 = "FILE";

            bool areEqualOrdinalCaseInsensitive = string.Equals(str1, str2, StringComparison.OrdinalIgnoreCase);
            Console.WriteLine("areEqualOrdinalCaseInsensitive=" + areEqualOrdinalCaseInsensitive);

            // change culture of the app to en-US (English, United States)
            Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
            bool areEqualUSCaseInsensitive = string.Equals(str1, str2, StringComparison.CurrentCultureIgnoreCase);
            Console.WriteLine("areEqualUSCaseInsensitive=" + areEqualUSCaseInsensitive);

            // change culture of the app to tr-TR (Turkey)
            Thread.CurrentThread.CurrentCulture = new CultureInfo("tr-TR");
            bool areEqualTRCaseInsensitive = string.Equals(str1, str2, StringComparison.CurrentCultureIgnoreCase);
            Console.WriteLine("areEqualTRCaseInsensitive=" + areEqualTRCaseInsensitive);
        }
    }
}
```

```
using System;
using System.Globalization;
using System.Threading;

namespace ConsoleApp2019
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            string str1 = "Strasse";
            string str2 = "Straße";

            // Ordinal equality check
            bool areEqualOrdinalCaseInsensitive = string.Equals(str1, str2, StringComparison.OrdinalIgnoreCase);
            Console.WriteLine("areEqualOrdinalCaseInsensitive=" + areEqualOrdinalCaseInsensitive);

            // Linguistic equality check in en-US (English, United States) culture
            Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
            bool areEqualUSCaseInsensitive = string.Equals(str1, str2, StringComparison.CurrentCultureIgnoreCase);
            Console.WriteLine("areEqualUSCaseInsensitive=" + areEqualUSCaseInsensitive);
        }
    }
}
```

Compare, equals, empty test – best practices

- Preferati metoda `string.Compare` pentru comparatii, in loc de `string.CompareTo`, intrucat va permite sa specificati modul in care realizati comparatia
- Utilizati metodele statice de comparare / test de egalitate pentru a evita potentiale probleme datorate string-urilor care pot fi null (remember: string is a reference type!)
- Front-end/presentation : user culture
- Backend :invariant culture

OOP

Object Oriented Programming

- Programare orientata pe obiect-

OOP

- o paradigma (filozofie) de programare
- axata pe conceptul de obiect,
 - o structura de date ce incapsuleaza (contine) atat date cat si logica (data and behaviour), Variabile si functii
- Obiectele si relatiile dintre ele incearca sa modeleze cat mai bine lumea reala
- Permite gestionarea mai usoara a complexitatii
- Codul devine modular (modularity)
- Codul devine refolosibil (reusability)
- Codul devine mentenabil (mentenability)
- Codul devine mai extensibil (extensibility)
 - Extensibilitatea- proprietatea unui sistem de a permite schimbari care vor afecat functionalitatea veche in mod minimal

OOP – clase si obiecte

- Clasa
 - Un tipar dupa dupa care **instantiem** obiecte
- Obiect
 - O instanta a unei clase
- In oop *totul** este un obiect
- Obiectele au *caracteristici* si *comportament*
- Exemplu de obiect
- Student
 - Caracteristici
 - Varsta
 - Nume
 - Prenume
 - Note
 - Comportamente
 - SePrezinta
 - CalculeazaMedia

• *aproape

OOP – clase si obiecte

- Exemplu de ~~obiect~~ ~~clasa~~
- Student
 - Date
 - Varsta
 - Nume
 - Prenume
 - Note
 - Comportamente
 - SePrezinta
 - CalculeazaMedia
- Clasa
 - Un tipar dupa dupa care **instantiem** obiecte

- Exemplu de obiect
- Student
 - Caracteristici
 - Varsta 23
 - Nume Popescu
 - Prenume Florin
 - Note 10, 7, 9, 10, 3
 - Comportamente
 - “Popescu Florin”
 - 7

Incapsularea – encapsulation

- Principiu OOP
- **Punerea impreuna a datelor si a comportamentelor** in cadrul unei singure entitati coerente si ascunderea detaliilor interne.
- **Punerea impreuna a campurilor si metodelor** care opereaza asupra acestora, in cadrul unui obiect si ascunderea datelor si campurilor si a metodelor interne, ascunderea implementarii.
- Pe romaneste
 - Punem impreuna variabile si functii care opereaza asupra variabilelor

OOP – clase – definire

- Cuvantul cheie “class”
 - Marcheaza inceputul unui bloc ce defineste o clasa
- Numele clasei
 - PascalCase
 - Sugestiv
- Membrii clasei (clas members)
 - Campuri – fields
 - Metode – methods
 - Constructori
 - Proprietati – properties +
- Obiect – instanta a clasei
 - Variabila in functie
 - Value type sau reference type?

```
/// <summary>  
/// Aici incepe definirea clasei  
/// </summary>  
0 references  
class Student {  
    ///  
    ///  
    /// continutul clasei va fi in acest block  
    ///  
    ///  
}  
///sfarsitul clasei
```

tipul obiectului	identificatorul obiectului
↓	↓
Student student = new Student();	

OOP – class – definire

- Cuvantul cheie “class”
 - Marcheaza inceputul unui bloc ce defineste o clasa
- Numele clasei
 - PascalCase
 - Sugestiv
- Membrii clasei (clas members)
 - Campuri – fields
 - Starea obiectului (state)
 - Metode – methods
 - Constructori
 - Proprietati – properties +
- Obiect – instantia a clasei
 - Variabila in functie
 - Reference type (*new*)
 - Default value?

```
/// <summary>
/// Aici incepe definirea clasei
/// </summary>
0 references
class Student {
    ///
    ///
    /// continutul clasei va fi in acest block
    ///
    ///
}
///sfarsitul clasei
```

tipul
obiectului identificatorul
obiectului

↓ ↓

```
Student student = new Student();
```

OOP – class fields

- Campurile clasei (**Class fields**)
 - Variabile la nivel de clasa
 - Orice tip, inclusiv alte obiecte!
 - camelCase , precum variabilele
 - Starea obiectului (state)
 - Campurile **sunt persistate**
 - Au aceeasi valoare pe toata durata existentei obiectului sau pana cand aceasta valoare va fi modificata
 - Reprezinta **starea** obiectului
- Definire
- Initializare

```
/// <summary>
/// Aici incepe definirea clasei
/// </summary>
2 references
class Student
{
    public string nume;
    public string prenume;
    public int varsta = 23;
    public int[] note;

    ///sfarsitul clasei
}
```

OOP- class methods

- Metode
 - Functii specific fiecarui obiect
 - Pot avea parametri, return type
 - Au acces la toate campurile
 - Dintr-o metoda a obiectului poate fi invocate orice alta metoda a obiectului!
 - PascalCase – functii

```
/// <summary>
/// Aici incepe definirea clasei
/// </summary>
2 references
class Student {
    public string nume;
    public string prenume;
    public int varsta = 23;
    public int[] note;

    0 references
    public void PrezintaTe()
    {
        Console.WriteLine($"Ma numesc {nume} { prenume} si am {varsta} de ani.");
    }

    0 references
    public double CalculeazaMedia() {
        double media = 0;
        foreach(var nota in note)
        {
            media += nota;
        }
        media = (double) media / note.Length;
        return media;
    }
}
///sfarsitul clasei
```

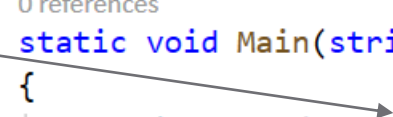

Initializarea, accesarea membrilor

- Obiect – instanța a clasei

- Reference type
- *new*

0 references

```
static void Main(string[] args)
{
    Student student = new Student();
}
```



- Accesarea membrilor obiectului

- “.”

- identificatorObiect.numeCamp
 - Trate ca oricare alte variabile

- identificatorObiect.NumeFuncție
 - Apeluri clasice de funcții

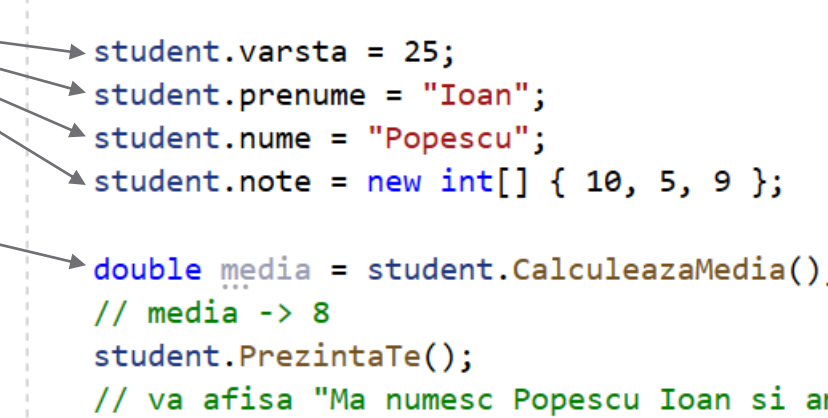
- NullReferenceException – când încercăm să accesăm un membru al unui obiect null.

0 references

```
static void Main(string[] args)
{
    Student student = new Student();

    student.varsta = 25;
    student.prenume = "Ioan";
    student.nume = "Popescu";
    student.note = new int[] { 10, 5, 9 };

    double media = student.CalculeazaMedia();
    // media -> 8
    student.PrezintaTe();
    // va afisa "Ma numesc Popescu Ioan si am 25 de ani."
}
```



OOP- constructor

- O functie a clasei
- Numele functiei = numele clasei
- Nu are tip returnat
- Folosit pentru initializarea unor variabile
- Poate primi parametri

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Student student = new Student();
        Console.WriteLine($"varsta studentului este {student.varsta} de ani");
        // va afisa "varsta studentului este 23 de ani"
        student.prenume = "Marin";
        student.num = "Chitac";
    }
}

4 references
class Student
{
    public string num;
    public string prenume;
    public int varsta;
    public int[] note = new int[10];

    1 reference
    public Student()
    {
        varsta = 23;
    }
}
```

Invocarea constructorului

Definirea constructorului

OOP- constructor

- Poate primi parametri
- Parametri sunt uzual folositi pentru a modifica campurile clasei

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Student student = new Student(25);
        Console.WriteLine($"varsta studentului este {student.varsta} de ani");
        // va afisa "varsta studentului este 25 de ani"
        student.prenume = "Marin";
        student.num = "Chitac";
    }
}
4 references
class Student
{
    public string num;
    public string prenume;
    public int varsta;
    public int[] note = new int[10];

    1 reference
    public Student(int varstaStudentului)
    {
        varsta = varstaStudentului;
    }
}
```

Invocarea constructorului



This

- Programul compileaza?
- Cine este cnp in interiorul setter-ului?
- Care va fi rezultatul executiei codului?

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Student student = new Student();

        student.SetCnp("321311");
    }
}
3 references
class Student {

    private string cnp;
    1 reference
    public void SetCnp(string cnp)
    {
        cnp = cnp;
    }
    0 references
    public string GetCnp()
    {
        return cnp;
    }
}
```

This

- This – cuvânt cheie (keyword)
 - Orice identificatory prefixat de “*this*” va face referinta la un membru al obiectului
- Ca urmare a executiei metodei “*Main*”, campul “*cnp*” al obiectului student va avea valoarea “321311”

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Student student = new Student();

        student.SetCnp("321311");
    }
}
3 references
class Student {

    private string cnp;
    1 reference
    public void SetCnp(string cnp)
    {
        this.cnp = cnp;
    }
    0 references
    public string GetCnp()
    {
        return cnp;
    }
}
```

Ordinea initializarii campurilor

- Ce va afisa programul?

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Student student = new Student(20);

        Console.WriteLine($"varsta studentului este {student.varsta} de ani");
    }
}
4 references
class Student
{
    public string nume;
    public string prenume;
    public int varsta = 25;
    public int[] note = new int[10];

    1 reference
    public Student(int varstaStudentului)
    {
        varsta = varstaStudentului;
    }
}
```

Ordinea initializarii campurilor

- Ce va afisa programul?
 - Rulam pas cu pas

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Student student = new Student(20);

        Console.WriteLine($"varsta studentului este {student.varsta} de ani");
    }
}

4 references
class Student
{
    public string nume;
    public string prenume;
    public int varsta = 25;
    public int[] note = new int[10];

    1 reference
    public Student(int varstaStudentului)
    {
        varsta = varstaStudentului;
    }
}
```

Ordinea initializarii campurilor

- 1) definirea si initializare
- 2) executia body-ului constructorului
- Va afisa "20"

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Student student = new Student(20);

        Console.WriteLine($"varsta studentului este {student.varsta} de ani");
    }
}
4 references
class Student
{
    1
    public string nume;
    public string prenume;
    public int varsta = 25;
    public int[] note = new int[10];

    1 reference
    public Student(int varstaStudentului)
    {
        2
        varsta = varstaStudentului;
    }
}
```


Initializarea campurilor obiectului

- In exteriorul obiectului
- Odata cu definirea
- In constructor sau functii

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Student student = new Student(25);
        Console.WriteLine($"varsta studentului este {student.varsta} de ani");
        // va afisa "varsta studentului este 25 de ani"
        student.prenume = "Marin";
        student.num = "Chitac";
    }
}
4 references
class Student
{
    public string num;
    public string prenume;
    public int varsta;
    public int[] note = new int[10];
    1 reference
    public Student(int varstaStudentului)
    {
        varsta = varstaStudentului;
    }
}
```

Campuri readonly

- [Link](#)
- Campurile marcate ca readonly nu pot fi modificate DUPA executia constructorului.
- Utile pentru valori specifice obiectelor care se doresc a fi imutabile
 - Ex: cnp, serieSasiu, etc....
- Pot fi modificate
 - La declarare si initializare
 - In constructori

```
// we can assign it as part of the constructor call
Student student = new Student("altCnp");
// we cannot assign it after the constructors execution
student.cnp = "dasdas";
}
}
7 references
class Student
{

    public readonly string cnp = "default cnp value";

    1 reference
    public Student(string cnp)
    {
        this.cnp = cnp;
        // we can assign it in constructor
    }

    0 references
    public void SetCnp(string newCnp)
    {
        this.cnp = newCnp;
        // we cannot assign it after the constructors execution
    }
}
```

Incapsularea

Ascunderea implementarii

Incapsularea – encapsulation

- Ascunderea implementarii – un alt aspect al incapsularii
- Punerea impreuna a datelor si comportamentelor in cadrul unei singure entitati coerente si **ascunderea detaliilor interne**.
- Punerea impreuna a campurilor si metodelor care opereaza asupra acestora, in cadrul unui obiect si ascunderea datelor si campurilor si a metodelor interne, **ascunderea implementarii**.
- Punerea impreuna a datelor – cu ajutorul claselor si a obiectelor
- Ascunderea implementarii – **modificatorii de acces**.

Modificatori de acces – private, public

- Public
 - membrii marcati ca si “public” sunt vizibili atat din metodele din interiorul clasei cat si din metodele sau functiile externe clasei
- Private
 - membrii marcati ca “private” sunt vizibili doar in interiorul metodelor clasei
 - Vizibili din obiectul caruia ii apartin
 - Vizibili din orice alt obiect instanta a aceeasi clase.

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Student student = new Student();

        string nume = student.nume;
        string cnp = student.cnp;
    }
}
3 references
class Student {
    public string nume;
    private string cnp;
    0 references
    public string GetCnp()
    {
        return cnp;
    }
}
```

eroare - de ce?

Modificatori de acces – private, public

- Public
 - membrii marcati ca si “public” sunt vizibili atat din metodele din interiorul clasei cat si din metodele sau functiile externe clasei
- Private
 - membrii marcati ca “private” sunt vizibili doar in interiorul metodelor clasei
 - Vizibili din obiectul caruia ii apartin
 - Vizibili din orice alt obiect instanta a aceeasi clase.

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Student student = new Student();

        string nume = student.nume;
        string cnp = student.cnp;
    }
}
3 references
class Student {
    public string nume;
    private string cnp;
    0 references
    public string GetCnp()
    {
        return cnp;
    }
}
```

cnp este marcat ca *private*

Accesul la membrii privati - Getter / Setter

- Private – accesibil doar din interiorul clasei
 - Asupra campurilor private pot opera toate metodele clasei indiferent de modificadorii de acces
- Getter / Setter
 - Getter method
 - Returneaza valoarea curenta a campului
 - Nume : `GetNumeCamp`
 - Setter
 - Permite modificare campului
 - Primeste un parametru de tipul campului
 - Returneaza void

```
class Student {  
    private string cnp;  
    1 reference  
    public string GetCnp()  
    {  
        return cnp;  
    }  
    0 references  
    private void SetCnp(string newCnp)  
    {  
        cnp = newCnp;  
    }  
}
```

Accesul la membrii privati - Getter / Setter

- Pot avea orice modificatori de acces
- Public getter, Private setter
 - Permite modificarea valorii doar din interiorul clasei
 - Controlul asupra valorii este strict in interiorul clasei
- Exercițiu:
 - Scrieti o clasa care sa modeleze un cont bancar. Contul va permite depunerea, extragerea numerarului precum si afisarea soldului.

```
class Student {  
  
    private string cnp;  
    1 reference  
    public string GetCnp()  
    {  
        return cnp;  
    }  
    0 references  
    private void SetCnp(string newCnp)  
    {  
        cnp = newCnp;  
    }  
}
```


Supraincarcare

Metode, constructori

Supraincarcarea - overloading

- [Link](#)
- Mai multe metode cu acelasi nume dar parametri diferiti.
- Metodele trebuie sa difere prin
 - Numarul parametrilor
 - Tipul parametrilor
 - Ordinea parametrilor
 - Compilatorul va alege metoda pe care sa o apeleze in functie de cele de mai sus!
- NU va compila daca difera doar prin
 - Numele parametrilor
 - Modificatori

Supraincarcarea - overloading

- Exemplu Math.Max
 - Ce fac cele 10 metode?

`Math.Max()`



`byte Math.Max(byte val1, byte val2)` + 10 overloads

Returns the larger of two 8-bit unsigned integers.

Returns:

Parameter `val1` or `val2`, whichever is larger.

CS1501: No overload for method 'Max' takes 0 arguments

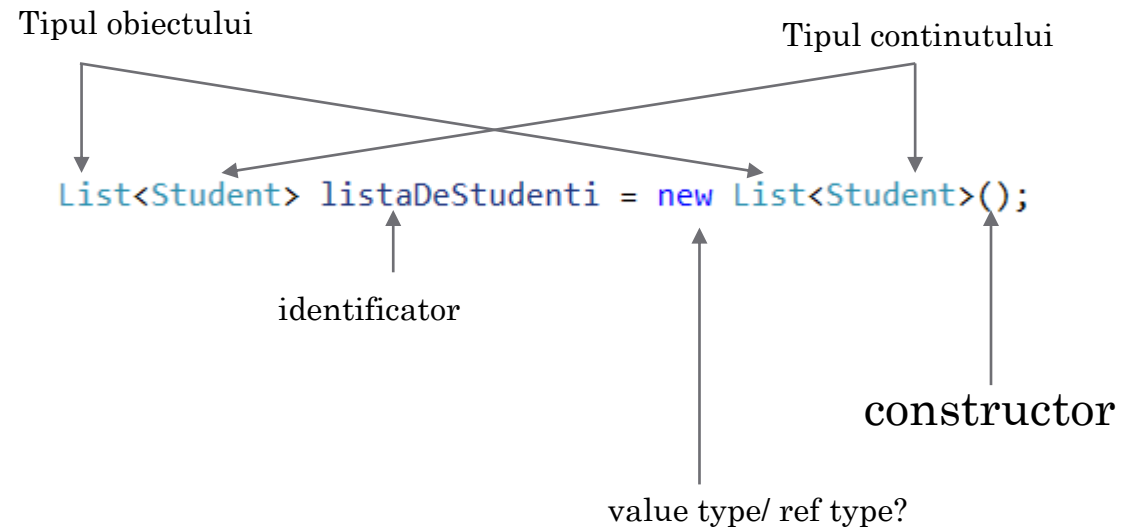
[Show potential fixes \(Ctrl+.\)](#)

Colecții

Clasa List

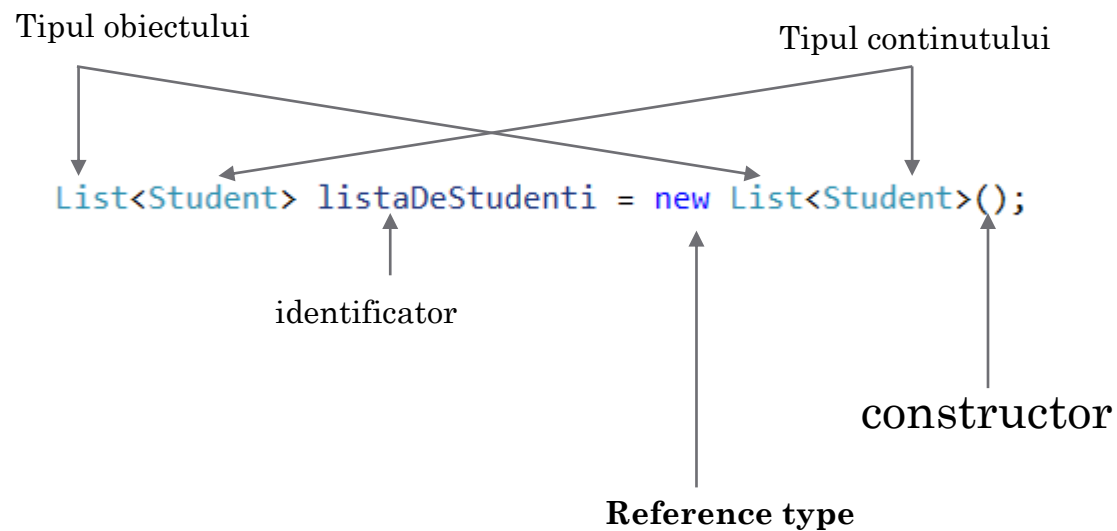
List

- List – clasa
 - listaDeStudenti – instanta a clasei “List” de tipul *Lista de Studenti*
- Lista generica
 - Poate contine orice tip de date
 - Datele continute trebuie sa fie toate de acelasi tip
- Valoarea default - ?



List

- List – clasa
 - listaDeStudenti – instanta a clasei “List” de tipul *Lista de Studenti*
- Lista generica
 - Poate contine orice tip de date
 - Datele continute trebuie sa fie toate de acelasi tip
- Valoarea default - **null**



List

```
// o lista goala de studenti
List<Student> listaDeStudenti = new List<Student>();

// initializeaza o lista copiind continutul colectiei (listei) date ca parametru
List<Student> clona = new List<Student>(listaDeStudenti);

// initializeaza o lista goala, setandu-i capacitatea initiala la 100 de elemente
List<Student> capacitateInitiala = new List<Student>(100);
```

- List.Capacity
 - numarul maxim de elemente pentru care s-a alocat memorie
 - default initial e 0
 - La aduagarea primului element Capacity creste la 4
 - Cand capacity este deposit, se realoca zona de memorie dubland capacitatea
 - Samd...
- List.Count
 - Numarul de elemente adaugate in lista

List

Adaugare in lista

```
private static void AdaugaUnStudent(Student deAdaugat, List<Student> listaDeStudenti)
{
    listaDeStudenti.Add(deAdaugat);
}
0 references
private static void AdaugaMultiStudenti(List<Student> deAdaugat, List<Student> listaDeStudenti)
{
    listaDeStudenti.AddRange(deAdaugat);
}
```

Parcurgerea listei

```
private static void ParcurgeLista(List<Student> listaDeStudenti)
{
    foreach (Student student in listaDeStudenti)
    {
        Console.WriteLine(student.Cnp);
    }
    for (var i = 0; i < listaDeStudenti.Count; i++)
    {
        Console.WriteLine(listaDeStudenti[i].Cnp);
    }
}
```

recomandat

List

- IMPORTANT
- Nu adaugați/eliminați elemente din lista in bucle for/foreach.
- Alternative
 - Folositi liste temporare pt a stoca referintele ce trebuie eliminate
 - Eliminati elementele IN AFARA buclelor buclei

```
//elimina un student pe baza referintei obiectului  
listaDeStudenti.Remove(student);
```

```
//elimina studentul de la pozitia 3  
listaDeStudenti.RemoveAt(3);
```

```
// elimina incepand cu pozitia 1 un numar de 5 studenti  
listaDeStudenti.RemoveRange(1, 5);
```

Va multumesc!