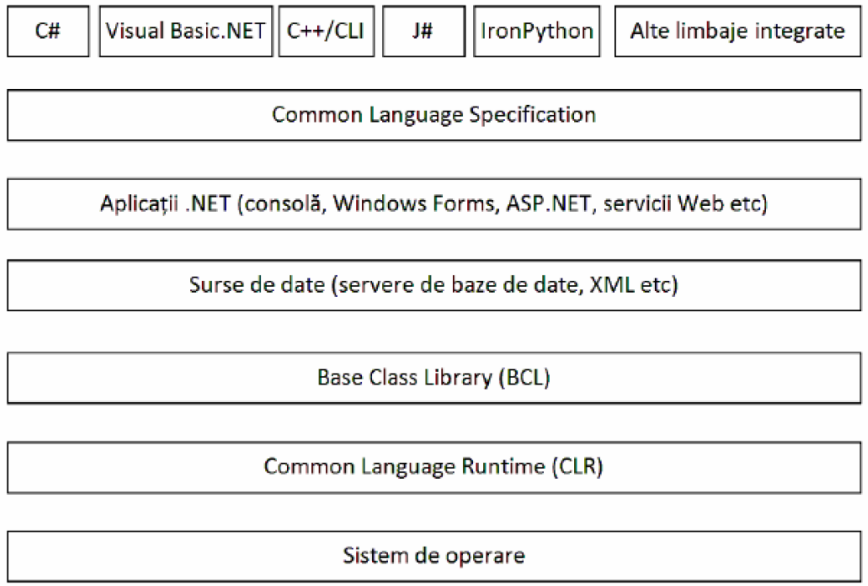
Metode avansate de programare

Informatică Româna

Curs 10

Platforma .Net

C# - Curs introductiv



Principalele caracteristici ale arhitecturii .NET

- Interoperabilitatea limbajelor
  - integrarea limbajelor astfel incat, programele, desi scrise in diferite limbaje, dar care sunt suportate de către platforma .NET, pot interopera
  - diversele componente ale unei aplicaţii pot fi scrise în limbaje diferite
- Portabilitate
  - Un program scris pentru platforma .NET poate rula fără nici o modificare pe orice sistem pe care platforma este instalată.

Common Intermediate Language (CIL)

- Toate limbajele, care sunt suportate de către platforma .NET, ex: C#, Managed C++, Visual Basic .NET, … , la compilare vor produce cod in acelasi limbaj intermediar: CIL (MSIL – IL);
- Asemanator cu bytecod-ul (java), CIL are trasaturi OO;
- CIL permite rularea aplicatiilor independent de platforma (cu aceeasi conditie ca la Java: sa existe o masina virtuala pentru acea platforma).

Common Language Runtime

- CLR este cea mai importanta componenta .NET Framework.
  - Este responsabila cu managementul si executia codului scris in limbajele .NET, aflat in format CIL;
- Este foarte similara cu Java Virtual Machine.
- In urma compilarii unei aplicatii, poate rezulta un fisier cu extensia .exe sau .dll (Dynamic Link Libraries) care va fi rulat de CLR.

Compilatorul C#

- Compilatorul C # compilează codul sursă specificat ca o multime de fisiere cu extensia .cs, într-un assembly (exe sau dll)
- O aplicatie de tip consola sau o aplicație Windows are o metoda Main (entry point) și este un .exe.
- O bibliotecă este un .dll și este echivalentă cu un .exe fără un punct de intrare (fara Main).
- Numele compilatorului C # este csc.exe
- Exemplu:

  > csc Test.cs

  Produce o aplicație numită Test.exe

```
using System;
   class Test
   {
       public static void Main()
       {
           Console.WriteLine(‘‘Hello world!’’);
       }
   }
```

Limbajul C#

- Deriva din C++ si are asemanari cu java
- Interoperabilitatea: esentiala la realizarea sistemelor software de dimensiuni mari
- Managementul automat al memoriei - *garbage collector*.

Limbajul C# - Exemplu

```java
//Java
public static void main(String[] args) {
    System.out.println("hello");
    launch(args);
}
```

```csharp
static void Main(string[] args)
{
   Console.WriteLine("this is just the begining!");
}
```

```csharp
static void Main() { }
static void Main(String[] args) { }
static int Main() { }
static int Main(String[] args) { }
```

Tipuri de date

- In C# toate tipurile de date sunt derivate din clasa System.Object
- Tipurile valoare: Contin date si sunt alocate pe stiva
  - tipurile simple (ex. char, int, float),
  - tipul enumerare (enum)
  - tipul structura (struct)
- Tipurile referinta: variabilele de acest tip stocheaza referinte catre obiectele care sunt alocate in heap.
  - interfata,
  - clasa
  - delegate
  - tablou

Tipuri valoare

- Toate tipurile valoare sunt derivate din clasa **`System.ValueType`**, care la randul ei este derivata din clasa **object** (alias pentru **System.Object**).
- Nu este posibil ca dintr–un tip valoare sa derivam alte tipuri.
- Atribuirea pentru un astfel de tip inseamna copierea valorii!!!
- Tipurile simple sunt identificate prin cuvinte rezervate, dar acestea reprezinta doar alias–uri pentru tipurile struct corespunzatoare din spatiul de nume **System**.

Corespondentele cu tipurile din spatiul de nume System

Cuvânt rezervat	Tipul alias
sbyte	System.SByte
byte	System.Byte
short	System.Int16
ushort	System.UInt16
int	System.Int32
uint	System.UInt32
long	System.Int64
ulong	System.UInt64
char	System.Char
float	System.Single
double	System.Double
bool	System.Boolean
decimal	System.Decimal

Exemple

- `int i = int.MaxValue; //constanta System.Int32.MaxValue`
- `string s = i.ToString(); //metoda System.Int32.ToString()`
- `string t = 3.ToString(); //idem`
- `double d = Double.Parse("3.14");`

Boxing si unboxing

```
System.Int32 a = 8;

object o = a; //boxing

int ua = (int)o; //unboxing
```

Constante

- O constanta este declarata folosind cuvantul rezervat const si trebuie inititalizata

```
const string Message = "Hello World";
```

- Java

```
final String message;
message="Hello World";
```

Tipul enumerat

```
public enum BorderSide { Left, Right, Top, Bottom }
BorderSide topSide = BorderSide.Top;
bool isTop = (topSide == BorderSide.Top); // true

public enum BorderSide : byte { //default int
    Left, Right, Top, Bottom
}

public enum BorderSide : byte {
    Left = 1, Right = 2, Top = 10, Bottom
}
```

BorderSide.Bottom = 11

Instructiuni conditionale si de control

- if …..
- switch
- while
- do … while
- for
- foreach `foreach (Student s in studentList) {…}`

Tablouri (arrays) - unidimensionale

- Declararea (spre deosebire de Java, nu se poate modifica locul parantezelor, nu se poate scrie: ~~int sir[].~~

```
int[] sir;
```

- Instantierea / initializare

```
sir = new int[10];
int[] a = new int[] {1,2,3};
```

sau in forma mai scurta:

```
int[] a = {1,2,3};
```

- Length = proprietate in clasa System.Array
- Orice tablou este un obiect, derivate din clasa System.Array.

Tablouri bidimensionale

```
int[] t1 = new int[2];

int[,] t2 = new int[3, 4];

int[,,] t3 = new int[5, 6, 7];

Console.WriteLine(t1.Rank.ToString() + t2.Rank + t3.Rank);
```

```
int[,] a = new int[3, 4] {
                {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
                {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
                {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
            };
```

Tablouri neregulate (jagged) array of array

```
int[][] scores = new int[5][];
for (int i = 0; i < scores.Length; i++)
{
      scores[i] = new int[4];
}

int[][] scores2 = new int[2][] { new int[] { 92, 93, 94 },
                                 new int[] { 85, 66, 87, 88 } };
Console.WriteLine(scores2.Rank);
```

Transmiterea parametrilor prin referinta: ref si out

- ref
  - Trebuie precizat si la definitia functiei si la apel
  - Argumentul poate fi de tip valoare sau de tip referinta
  - Parametrul trebuie sa fie initializat
- out
  - Trebuie precizat si la definitia functiei si la apel
  - Cauzeaza transmiterea prin referinta
  - Nu impune intializarea paramentrului actual

Exemplu ref

```
class NumberManipulator
{
        public void Swap(ref int x, ref int y)
        {
            int temp; temp = x; x = y; y = temp;
        }
        static void Main(string[] args)
        {
            NumberManipulator n = new NumberManipulator();
           int a = 100;
          int b = 200;
           n.Swap(ref a, ref b);
        }
    }
```

Exemplu out

```
class NumberManipulator
    {
        public void Init(out int x, out int y)
        {
            x = 5;
            y = 9;
        }
        static void Main(string[] args)
        {
            NumberManipulator n = new NumberManipulator();
            int a;
            int b;
            n.Init(out a, out b);
        }
    }
```

params

- Permite un numar variabil de parametri
- Parametri actuali pot fi:
  - o lista de parametrii separati prin virgula
  - un tablou.

```
public class MyClass
   {
       public static void UseParams(params int[]
       list)
       {
           for (int i = 0; i < list.Length; i++)
           {
               Console.Write(list[i] + " ");
           }
           Console.WriteLine();
       }
       static void Main()
       {
           UseParams(1, 2, 3, 4);
           UseParams();
           int[] myIntArray = { 5, 6, 7, 8, 9 };
           UseParams(myIntArray);
       }
   }
```

Clase C#

- Clasele reprezinta tipuri referinta.
- O clasa poate sa mosteneasca o singura clasa si poate implementa mai multe interfete.

C# Clase

```
class Person
    {
        //fields
        //methods
        //properties
    }
```

```
Field declaration:
[static] [access modifier] [new][readonly] Type field_name[= init_val];
```

Atribuirea unei valori unui camp de tip `readonly` se poate face la declararea sa, sau prin intermediul unui constructor.

Nu este obligatoriu cunoasterea valorii unui astfel de camp la compilare.

Constructor /Destructor

- Constructor

```
[access modifier] ClassName ([list of parameters]){…}
```

- Constructorul poate fi supraincarcat
- Daca nu este definit nici un constructor, compilatorul genereaza un constructor default
- **Destructor**

```
~ClassName(){…}
```

- Destructorul este apelat automat când obiectul urmeaza să fie distrus de catre Garbage Collector

Metode

```
[access_m] [static][inheritance_m] Type method_name(list_params )
{
                //body of the method
}

list_params: [modifier] type1 param1, [modifier] type2 param2,…

Modifier: ref, out
```

Modificatorii de acces

private	Access only within the same class
protected internal	Access only from the same project or from subclasses
internal	Access only from the same project
protected	Access from subclasses
public	No restriction

Valori implicite:

- fields/methods: private
- classes, interfaces: internal

Properties

Properties look like fields from the outside but act like methods on the inside.

```
class Person
{
        private string name;
        public string Name
        {
            get
            {
                return name;
            }
            set
            {
                name = value;
            }
        }
}

Person p = new Person();
p.Name = "Andu"; //set
string nume = p.Name; //get
```

```
class Person
    {
        //String nume;
        public string Nume
        {
            get; set;
        }
    }


    //Java
    class Person{
        private String name;

        public String getName() {return name; }
        public void setName(String name) {
            this.name=name;
        }
    }
```

Indexatori

- Un indexator este o generalizare a supraıncarcarii operatorului [] din C++.

```
class MyVector
   {
        private double[] v;
        public MyVector(int length)
        {
            v = new double[length];
        }
        public int Length
        {
            get
            { return v.Length; }
        }
        public double this[int index]
        {
            get
            { return v[index]; }
            set
            { v[index] = value; }
        }
   }
```

```
class Test
   {
       static void Main()
       {
           MyVector v = new MyVector(10);
           v[0] = 0;
           v[1] = 1;
           for (int i = 2; i < v.Length; i++)
           {
               v[i] = v[i - 1] + v[i - 2];
           }
           for (int i = 0; i < v.Length; i++)
           {
               Console.WriteLine("v[" +i.ToString() + "]
                        =" +v[i]);
           }
       }
   }
```

Mostenirea

- Mostenire simpla de clasa si implementare multipla de interfata!!!!

```
class Person{ … }
class Student : Person { … }

class A { … }
interface B{…}
interface C : B {…} //mostenire de interfata

interface CC {…}
class SA : A, B, CC { … } //mostenire de clasa,
          //implementare de interfata
```

Metode virtuale

```
public class Casa:Imobil
   {
       public double SuprafataTeren { set; get; }
       public Casa(string adresa, double supCasa, double
                  supTeren) : base(adresa, supCasa)
       {
           this.SuprafataTeren = supTeren;
       }

       public override double Pret()
        //public new double Pret()
       {
           return base.Pret() + SuprafataTeren;
       }
       public override string ToString()
       {
           return base.ToString()+" "+SuprafataTeren;
       }
   }
```

```
public class Imobil
   {
       public double Suprafata { set; get; }
       public string Adress { set; get; }
       public Imobil(string adress, double suprafata)
       {
           this.Adress = adress;
           this.Suprafata = suprafata;
       }
       public virtual double Pret()
       {
           return 100 * Suprafata;
       }
       public override string ToString()
       {
           return Adress + " " + Suprafata;
       }
   }
```

```
Imobil[] v = new Imobil[3];
v[0] = new Casa("cj", 100, 500);
v[1] = new Imobil("cj", 100);
Console.WriteLine(v[0].Pret());
Console.WriteLine(v[1].Pret());
```

Cuvantul base

- Apel constructor din clasa de baza:

```
public class Casa:Imobil
{
      public Casa(string adresa, double supCasa, double supTeren) : base(adresa,
supCasa)
}
```

- Apel metoda din clasa de baza:

```
public override string ToString()
{
     return base.ToString()+" "+SuprafataTeren;
}
```

Cuvantul sealed

- sealed method = nu mai poate fi suprascrisa (final Java)

- sealed class = nu putem deriva alte clase, previne subtipizarea

static

- Static constructor - initialization
- Static classes
  - A class can be marked **static**, indicating that it constains only static members (fields, methods, properties).

Operatorii as and is

- Operatorul as executa un downcast care se evalueaza la null in cazul in care nu se poate face downcastul

```
Person pers = new Person();
Student st = pers as Student(); //st is null
Person pers1 = new Student();
Student st1 = pers1 as Student(); //st is not null
```

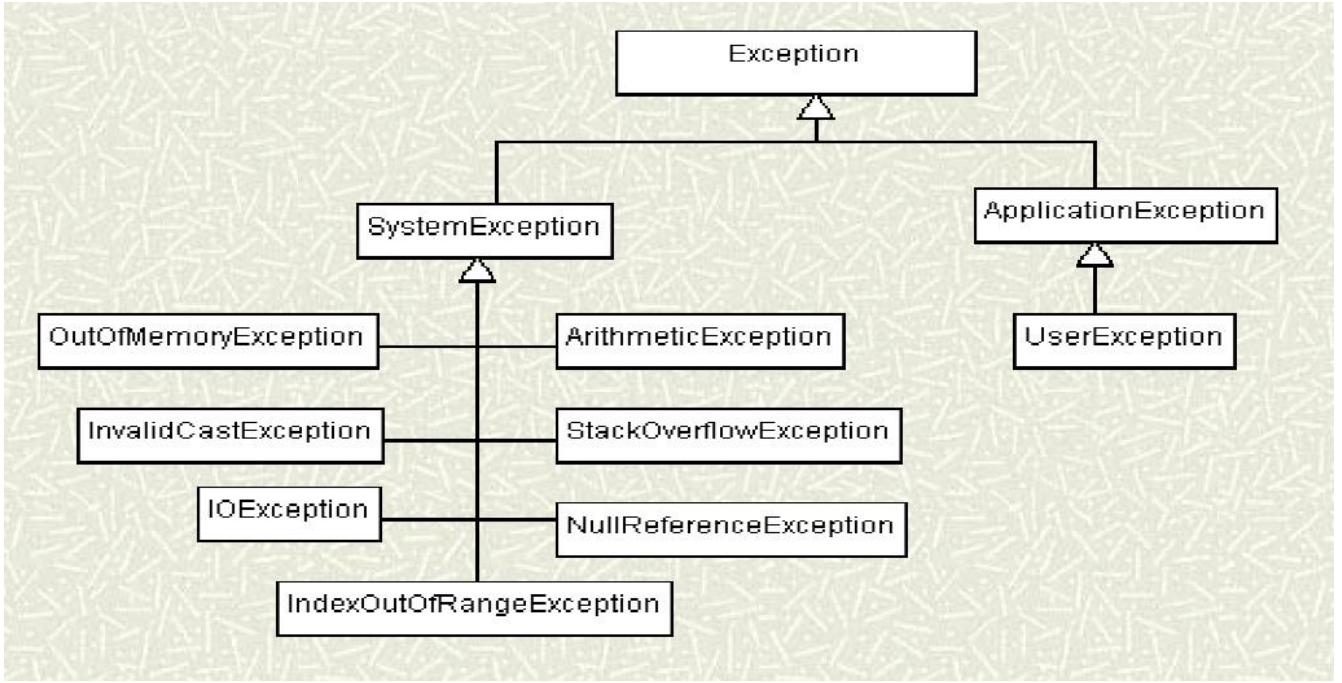
- Operatorul is verifica daca un downcast s-a finalizat cu success. Se foloseste de obicei inainte de a face downcastul.

```
if ( pers1 is Student){ … }
```

string

- Imutable
- Clasa **System.String** (pentru care se poate folosi aliasul "**string**“)
  - Operatorii "==" ¸si "!="
  - Clasa String pune la dispozitie metode pentru:
    - comparare : Compare, CompareOrdinal, CompareTo,
    - Cautare : EndsWith, StartsWith, IndexOf, LastIndexOf,
    - modificare (=obtinerea altor obiecte pe baza celui curent)
    - Concat, CopyTo, Insert, Join, PadLeft, PadRight, Remove, Replace, Split,
    - Substring, ToLower, ToUpper,Trim, TrimEnd, TrimStart
    - Accesarea unui caracter aflat pe o pozitie i a unui sir s: s[i].
    - Split()
  - String este un tip referinta!

Exceptii



- All exceptions in C# are runtime exceptions.
- There is no equivalent to Java's compile-time checked exceptions.

Cursul urmator

- Generics
- Delegates
- Events
- DataStructures
- Lambda