

1. The .NET Framework

Sastoji se iz dva dela:

- a. **Common Language Runtime (CLR)** – okruženje u kojem se program izvršava.
- b. **.NET Framework class libraries** – biblioteke koje sadrže sve funkcije neophodne da se kreira program u .Net okruženju.

2. CLR Projekat

Svaki CLR program se sastoji od jednog ili više *assembly*, koji se sastoji od više programskih jedinica i resursa a koji formiraju jednu funkcionalnu jedinicu (*Assemblies are self- describing installation units, consisting of one or more files*).

- a. **Primer01.**
- b. `using namespace` – sve programske jedinice (sav kod) u .Net arhitekturi su realizovane u okviru imenskih prostora. Standardne jedinice (one koje se najčešće koriste su definisane u okviru **System** imenskog prostora).
- c. Deklarisanje imenskog prostora:

```
namespace mojProstor
{
    // kod...
}
```

- d. Da bi se tačno odredila klasa koja se koristi mora se definisati putanja do nje. Postoje dva načina kako se putanja definiše:

1. Puna putanja do namespace i klase/promenljive/metode u njoj (*Supplying the Fully Qualified Namespace*):

```
System.Windows.Forms.MessageBox.Show("Zdravo");
```

2. Upotrebom **using** direktive:

```
using System.Windows.Forms;
...
MessageBox.Show("Zdravo");
```

3. Podaci i promenljive

.Net okruženje proširuje standardni C++:

- a. Svi fundamentalni ISO/ANSI tipovi se mogu koristiti. Net okruženje razlikuje dve vrste tipova podataka: primitivne tipove (**value types**) i referentni tipovi (**reference types**), koji se razlikuju po načinu manipulacijama vrednostima u memoriji. Osnovni tipovi manipulišu samom vrednošću dok referentni tipovi manipulišu referencama vrednosti.
- b. **Predefinisani C# tipovi podataka.** Svi tipovi podataka u **.Net** arhitekturi su klase tako da deklarisanje promenljivih zapravo znači kreiranje instanci klasa, pa se oni mogu ponašati i kao obični tipovi podataka i kao objekti (zavisi od situacije).

Sledeća tabela sadrži tipove podataka u **C#** programu:

Primitivni tip podatka	Veličina (bajt)	.Net tip podatka	Vrednost
bool	1	System.Boolean	true / false
char	1	System.Char	16-bitni (Unicode) karakter
sbyte	1	System.SByte	-128 : 127
byte	1	System.Byte	0 : 255
short	2	System.Int16	-32,768 : 32,767
ushort	2	System.UInt16	0 : 65,535
int	4	System.Int32	-2,147,483,648 : 2,147,483,647

uint	4	System.UInt32	0 : 4,294,967,295
long	8	System.Int64	-9,223,372,036,854,775,808 : 9,223,372,036,854,775,807
ulong	8	System.UInt64	0 : 18,446,744,073,709,551,615
float	4	System.Single	$1.5 \cdot 10^{-45}$ to $3.4 \cdot 10^{38}$
double	8	System.Double	$5.0 \cdot 10^{-324}$ to $1.7 \cdot 10^{308}$
decimal	16	System.Decimal	$1.0 \cdot 10^{-28}$ to $7.9 \cdot 10^{28}$
Referentni tip podatka	Veličina (bajt)	.Net tip podatka	Vrednost
Object		System.Object	Osnovni tip iz kojeg su svi drugi izvedeni
string		System.String	Niz Unicode karaktera

Svaka od datih klasa proširuje karakteristike osnovnih tipova podataka (npr. **ToString()** metoda). Kada je potrebno kompajler vrši automatsku konverziju iz osnovnog tipa u odgovarajuću klasu (**boxing/unboxing**).

Kada postoji mogućnost konfuzije između int tipova podataka, kompajler svodi na osnovni int. Da bi se to izbeglo, potrebno je eksplicitno definisati koji je int tip u pitanju, kao u primerima:

```
uint ui = 1234U;
long l = 1234L;
ulong ul = 1234UL;
```

- c. Transformacija podataka iz jednog tipa u drugi (**Casting**) se realizuje na implicitni (kada kompajler automatski izvrši transformaciju podatka) ili eksplicitni način. Eksplicitni način se koristi kada postoji mogućnost da dođe do gubitka podataka ili kada se tipovi ne mogu jednostavno konvertovati (string u int i sl.).

Sledeći transformacije među tipovima se mogu implicitno realizovati:

sbyte	short, int, long, float, double, decimal
byte	short, int, long, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
float	double
double	decimal
char	ushort, int, uint, long, ulong, float, double, decimal

Za eksplicitno konvertovanje podataka koriste se metode **Convert** klase:

Metoda	Tip u koji se konvertuje
ToBoolean(izraz)	bool
ToByte(izraz)	byte
ToChar(izraz)	char
ToDateTime(izraz)	DateTime
ToDecimal(izraz)	decimal
ToDouble(izraz)	double
ToInt16(izraz)	16-bit signed integer
ToInt32(izraz)	integer
ToInt64(izraz)	long
ToSByte(izraz)	sbyte
ToSingle(izraz)	Single

ToString(<i>izraz</i>)	string
ToUInt16(<i>izraz</i>)	16-bit unsigned integer
ToUInt32(<i>izraz</i>)	32-bit unsigned integer
ToUInt64(<i>izraz</i>)	64-bit unsigned integer

Metode se koriste tako što se prosleđuje promenljiva (*izraz*) koji se konvertuje u odgovarajući tip. Rezultat metode je konvertovana vrednost.

- d. Konstante se definišu pomoću rezervisane reči **const**.

```
const int ui = 1234567;
```

- e. Enumeracija se definiše pomoću rezervisane reči **enum**.

```
enum Dani {Pon, Uto, Sre, Cet, Pet, Sub, Ned};
```

U prethodnom primeru Pon dobija vrednost 0, Uto 1 i td. Članovima enumeracije se mogu dodeliti i konkretne vrednosti:

```
enum Dani {Pon=1, Uto, Sre, Cet, Pet, Sub, Ned};
```

Članu enumeracije se pristupa navođenjem naziva enumeracije:

```
int x = (int) Dani.Pon;
```

4. Nizovi i stringovi

Dinamičko zauzimanje memorije je drukčije u CLR programima nego u standardnom C++. CLR održava sopstvenu memoriju, CLR memoriju (CLR memory heap). Ne postoji potreba za direktnim upravljanjem memorijom. Memorijom upravlja garbidž kolektor (**garbage collector**) koji oslobađa memoriju zauzetu promenljivama koje se više ne referenciraju u programu. On ujedno vrši i sažimanje memorije (reorganizaciju podataka u memoriji).

- a. Kao što je već rečeno, postoje dve vrste promenljivih – value types (čije se vrednosti smeštaju na stek) i reference types (čije se vrednost smeštaju na heap, dok se na steku čuvaju samo njihove reference). Oslobađanje memorije zauzete value types promenljivama se vrši kada one izađu iz dosega, dok memoriju zauzetu reference types promenljivom oslobađa garbidž kolektor kada sve njegove reference izađu iz dosega.

Nova reference type instanca se kreira pozivom **new** operatora. Pristup atributima i metodama, kao i namespace dosezima, se realizuje pomoću **.** operatora.

- b. **CLR nizovi** – sadrže dodatne opcije u odnosu na standardne C++ nizove (pošto su i oni sami realizovani kao klasa). Nizovi su realizovani kao reference type objekti tako da se moraju kreirati pomoću **new** operatora (mora se kreirati referenca na niz i rezerverisati memorija). O memoriji koju zauzimaju brine garbidž kolektor, tako da svaki niz mora biti referencira. Nizovi se deklarishu tako što se iza tipa niza navodi prazna uglasta zagrada []. Svi elementi niza su istog tipa.

Jednom deklarisanu promenljivu tipa niz može ukazivati na bilo koji niz istih dimenzija čiji su elementi istog tipa. To znači da je u slučaju:

```
int[ ] data;
```

promenljiva **data** referencera na neki jednodimenzionalni niz integer vrednosti pri čemu taj niz nije konkretno definisan. Da bi **data** promenljiva referencirala konkretni niz, on se prvo mora inicijalizovati pomoću **new** operatora.

```
int[ ] data = new int[100]; // niz koji sadrži 100 int promenljivih
```

Moguće je deklarishati više referenci na isti niz.

```
int[ ] copy = data;
```

Moguće je deklarishati referencu na niz kojoj se dinamički dodeljuje niz.

```
int[ ] data;
```

```
data = new int[10];
```

Takođe je moguće promeniti referencu na novi niz:

```
data = new int[45];
```

Ukoliko na prethodni niz ne postoji više ni jedna reference, on se briše iz memorije.

Elementi niza se mogu eksplicitno definishati:

```
double[ ] s = { 3.3, 1.3, 9.8, 6.2, 5.8, 4.1, 7.3, 8.6};
```

Elementi niz mogu biti bilo kog tipa:

```
String[ ] Imena = { "Vlada", "Slavica", "Branko", "Dragan", "Dinu"};
```

Nizovi su indeksirani od nule, 0. Elementima niza se može pristupati na više načina:

```
data[0] = 2;
```

```
...
```

```
data[44] = 1234;
```

Elementima niza se može pristupati preko iteratora:

```
for(int i = 0 ; i < data.Length ; i++)
```

```
data[i] = 2*(i+1);
```

ili

```
foreach(int item in data)
```

```
item = 2*item + 1;
```

c. **Multidimenzionalni nizovi** – se definišu na sledeći način:

tip[,] promenljiva za dvodimenzionalni niz, **tip[, ,] promenljiva** za trodimenzionalni niz i sl. Maksimalna dozvoljena dimenzija niza koja je podržana u CLR okruženju je 32. dvodimenzionalni niz se može definisati na sledeći način:

```
int[ , ] data = new int[10, 10];
```

d. **Primer02.**

e. **Tip String** – klasa **String** je definisana u **System** imenskom prostoru. Ona reprezentuje niz čiju su elementi tipa **System.Char** (*Unicode*). String se može kreirati na sledeći način:

```
string izjava = "Nisam kriv.";
```

String promenljive je moguće dodavati uz pomoć operatora +:

```
string ime1 = "Vlada";
```

```
string ime2 = "Vladica";
```

```
string izjava = ime1 + " i " + ime2;
```

Za modifikovanje se može koristiti više funkcija koje su metode klase String:

- String trimming: **Trim()**, **TrimStart()**, **TrimEnd()**.
- String padding: **PadLeft()** and **PadRight()**.
- Converting string to lower or upper case: **ToUpper()** and **ToLower()**.
- Inserting a string at a given position in an existing string: **Insert()**.
- Replacing all occurrences of a given character in a string with another character or all occurrences of a given substring with another substring: **Replace()**.

Operacije za pretraživanje stringova:

- Testing whether a string starts or ends with a given substring: **StartsWith()** and **EndsWith()**.
- Searching a string for the first occurrence of a specified character or a substring: **IndexOf()**. Function returns the index if it is present or -1 if it is not found.
- Searching a string for the first occurrence of a specified character or a substring from end of the string: **LastIndexOf()**. Function returns the index if it is present or -1 if it is not found.

f. **Primer03.**

5. Funkcije

Uglavnom se ne razlikuju od funkcija u standardnom C++ okruženju. Ukoliko se želi neki parametar koji nije referenca proslediti kao referenca mora se ispred njega dopisati rezervisana reč **ref** kao u primeru:

```
public void MyMethod ( ref int i) { i = 10; }
```

6. Grananje i iteracija toka programa

Ne razlikuje se od standardnog C++ okruženja.

a. IF-THEN:

```
if (<test>){  
    <code executed if <test> is true>;  
}  
else {  
    <code executed if <test> is false>;  
}
```

b. SWITCH:

```
switch (<testVar>) {  
    case <comparisonVal1> : <code to execute if <testVar> == <comparisonVal1>>  
                           break;  
    case <comparisonVal2> : <code to execute if <testVar> == <comparisonVal2> >  
                           break;  
    ...  
    case <comparisonValN> : <code to execute if <testVar> == <comparisonValN> >  
                           break;  
    default :              <code to execute if <testVar> != comparisonVals>  
                           break;  
}
```

c. DO-WHILE:

```
do {  
    <code to be looped>  
} while (<Test>);
```

d. WHILE:

```
while (<Test>)  
{  
    <code to be looped>  
}
```

e. FOR:

```
for (<initialization>; <condition>; <operation>)  
{  
    <code to loop>  
}
```

7. Strukture i klase

CLR okruženje omogućuje rad sa strukturama i klasama, pri čemu su strukture value type tipovi dok su klase reference type tipovi. Ne postoje friend funkcije.

- a. **Klase (class)** – trebala bi biti klasa jednostavnog tipa (namenjena realizaciji novih primitivnih tipova). Promenljive ovog tipa su reference (smeštene na steku) dok se podaci kreiranog objekta smestaju na hipu. Deklariše se sa **class** naredbom.

```
class Kocka {  
    // konstruktori  
    public Kocka() {  
        Duzina = 1.0;  
        Sirina = 1.0;  
        Visina = 1.0;  
    }  
    public Kocka(double a_d, double a_s, double a_v) {  
        Duzina = a_d;
```

```

        Sirina = a_s;
        Visina = a_v;
    }

    // Zapremina kocke
    public double Zapremina(){
        return Duzina*Sirina*Visina;
    }

    // atributi klase
    private double Duzina; // Duzina kocke
    private double Sirina; // Sirina kocke
    private double Visina; // Visina kocke
};

```

Kreiranje instance klase Kocka se vrši na sledeći način:

```

Kocka mKocka; // pokazivač na tip Kocka
Kocka novaK = new Kocka(10,20,30);
mKocka = new Kocka(); // poziv predefinisnog konstrukora

```

b. Primer04.

- c. Struct** – je po svojim karakteristikama slična klasi – može imati svoje private/public attribute, konstruktor i metode. Razlikuje se po tome što su instance struktura value type. Ne kreira se predefinisani konstruktor. Deklariše se sa **struct** naredbom.

```

struct Dimenzije {
    public double duzina;
    public double sirina;
}

```

Instance strukture se mogu definisati na sledeći način:

```

Dimenzije tacka;
tacka.duzina = 10.2;
tacka.sirina = 11.1;

```

Strukture se ne mogu nasleđivati.

d. Primer05.

- e. Class Properties** – omogućuje poziv određenih metoda klase kao da su atributi. Property element klase se razlikuje od običnog atributa u tome da on ne definiše lokaciju u memoriji, nego kada se njemu pristupa, poziva se određena metoda. Svaki property element može sadržati **get** i/ili **set** metodu. Kada se identifikator property elementa nađe sa leve strane operatora dodele (prima vrednost), tada se zapravo poziva njegova set metoda, a kada se koristi za pristup vrednosti, poziva se njegova get metoda. Property element može biti write-only (sadrži samo set metodu) ili read-only (sadrži samo get metodu).

Skalarni property element ima samo jednu vrednost. Definiše se na sledeći način:

```

public <type> <PropertyName>
{
    get { ... };
    set { ... };
}

```

Kao u primeru:

```

private double visina;
private double sirina;

public double Visina {

```

```

        set { visina = value; }
        get { return visina; }
    }
    public double Sirina {
        set { sirina = value; }
        get { return sirina; }
    }
    public double Povrsina
    {
        get { return visina*sirina; }
    }
}

```

Dozvoljeno je definisati operatore pristupa za get/set metodu (public/private/protected).

Indeksirani property element se koriste za pristup atributu neke klase putem operatora uglastih zagrada []. Na osnovu indeksa zadatog između zagrada se određuje kojem se elementu pristupa. Indeksirani property element se naziva Indexier a pristupa mu se preko naziva klase. Indeksirani property element se definiše na sledeći način:

```

<modifier> <return type> this [argument list] {
    get {
        ...
    }
    set {
        ...
    }
}

```

Kao u primeru:

```

class MyClass {
    private string[] data = new string[5];
    public string this[int index] {
        get {
            return data[index];
        }
        set {
            data[index] = value;
        }
    }
}

```

Primer upotrebe indeksiranog property elementa:

```

MyClass mc = new MyClass();
mc[0] = "Mira";
mc[1] = "Slavica";
mc[2] = "Srdjan";
mc[3] = "Miroslav";
mc[4] = "Dinu";

Console.WriteLine("{0}, {1}, {2}, {3}, {4}", mc[0], mc[1], mc[2], mc[3], mc[4]);

```

f. **Primer06, Primer07.**

g. **Static funkcije, klase, metode property** – definišu se isto kao i obični elementi sa tom razlikom da se poziva na nivou čitave klase a ne na nivou instance objekta.

h. **Operator overloading in Value Classes** – je slično kao i u standardnom C++. Preklapanje operatora je opisano na primeru sabiranja dva kompleksna broja (pretpostaviti da postoji definisana klasa kompleksnih brojeva):

```

public static Complex operator + (Complex iZ, Complex uZ)
{
    Complex resZ = new Complex( );
    resZ.real = iZ.real + uZ.real;
    resZ.imag = iZ.imag + uZ.imag;
    return resZ;
}

```

i. **Primer08.**

- j. **Nasleđivanje i virtualne funkcije** – sve klase u CLR okruženju su izvedene klase pošto nasleđuju **System.Object** klasu. C# dozvoljava nasleđivanje klase i interfejsa. Dozvoljeno je nasleđivanje samo jedne klase, dok je moguće naslediti više interfejsa.

Ako klasa sadrži čistu virtuelnu funkciju (pure virtual function), onda se data klasa mora označiti ključnom rečju **abstract** kao apstraktna klasa. Apstraktne metode se označavaju takođe ključnom rečju **abstract** na kraju svoje deklaracije.

Prilikom redefinicije funkcije (svejedno da li je virtualna ili ne) mora se upotrebiti ključna reč **override**.

Ako klasa naslednica ne redefiniše sve apstraktne metode osnovne klase, tada je i ona apstraktna i mora sadržati ključnu reč **abstract**.

Ukoliko se želi dozvoliti reimplementacija metode u bilo kojoj klasi naslednici, onda se ona proglašava virtuelnom.

U nastavku sledi primeri

```

abstract class Container {
    // Function for calculating a volume - no content This is defined as an 'abstract' virtual
    // function, indicated by the 'abstract' keyword
    public abstract double Volume();

    // Function to display a volume
    public virtual void ShowVolume() {
        Console.WriteLine("Volume is {0}", Volume());
    }
}

```

```

class Box : Container {                                     // Derived class
    // Constructor
    public Box() {
        m_Length = 1.0;
        m_Width = 1.0;
        m_Height = 1.0;
    }
    // Constructor
    public Box(double lv, double wv, double hv) {
        m_Length = lv;
        m_Width = wv;
        m_Height = hv;
    }
    // Function to show the volume of an object
    public override void ShowVolume() {
        Console.WriteLine("Box usable volume is {0}", Volume());
    }
    // Function to calculate the volume of a Box object
    public override double Volume() { return m_Length*m_Width*m_Height; }
}

```



```

        protected double m_Length;
        protected double m_Width;
        protected double m_Height;
    }

    class GlassBox : Box { // Derived class
        // Constructor
        public GlassBox(double lv, double wv, double hv)
            : base(lv, wv, hv) // način da se pristupa klasi koja se nasleđuje
        {
        }
        // Function to calculate volume of a GlassBox allowing 15% for packing
        public override double Volume() { return 0.85 * m_Length * m_Width * m_Height; }
    }

```

Pregled karakteristika nasleđivanja u CLR okruženju:

1. Nasleđivanje je uvek **public**.
2. Funkcija koja nema definiciju je apstraktna i mora biti označena ključnom rečju **abstract**.
3. Klasa koja sadrži apstraktne metode, mora biti kvalifikovana kao apstraktna.
4. Klasa se može deklarirati kao apstraktna čak i kada nema apstraktnih metoda (tada ne mogu postojati instance date klase).
5. Ključna reč **override** se mora eksplicitno navesti kada se vrši redefinicija metode.

k. Primer09.

- l. Interface klase (Interface Classes)** – definišu interfejs koje treba da implementiraju druge klase. Obezbeđuju standardni pogled na određenu operaciju i na ono što se treba realizovati. Interfejs klase ne implementiraju svoje metode, to čine klase koje ih koriste. Interfejs klase se deklariraju uz pomoć ključne reči **interface**. Sve metode definisane u interfejs klasi su javne (**public**).

U nastavku sledi primer interfejs klase **IContainer**:

```

public interface IContainer
{
    double Volume();
    void ShowVolume();
};

```

Svaka klasa koja implementira **IContainer** interfejs mora da implementira sve njegove metode.

m. Primer10.

- n. Destruktori – Finalizers funkcije u Reference klasam** – u CLR okruženju je moguće definisati destruktora za klasu preko **finalizer** funkcije. U C# je to posebna metoda klase koju poziva garbidž kolektor prilikom oslobađanja memorije koju je zauzela instanca klase. Ona je deo **System.Object** klase i mora se reimplementirati u klasi koja se stvara. Ova metoda se koristi u slučajevima kada je potrebno osloboditi neke resurse koje garbidž kolektor ne može osloboditi (npr. otvorena datoteka, konekcija ka bazi i sl.). Finalizer metoda isto kao i destruktora u C++, ne vraća vrednost i ne prima parametre. U nastavku sledi primer za finalizer funkciju.

```

protected override void Finalize() {
    try {
        // destructor implementation
    }
}

```

```
        finally {  
            base.Finalize();  
        }  
    }  
}
```