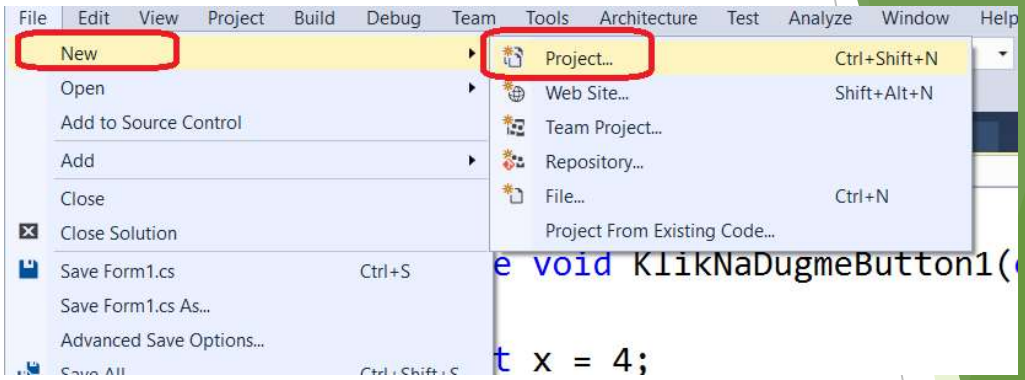
Osnove programskog jezika C#

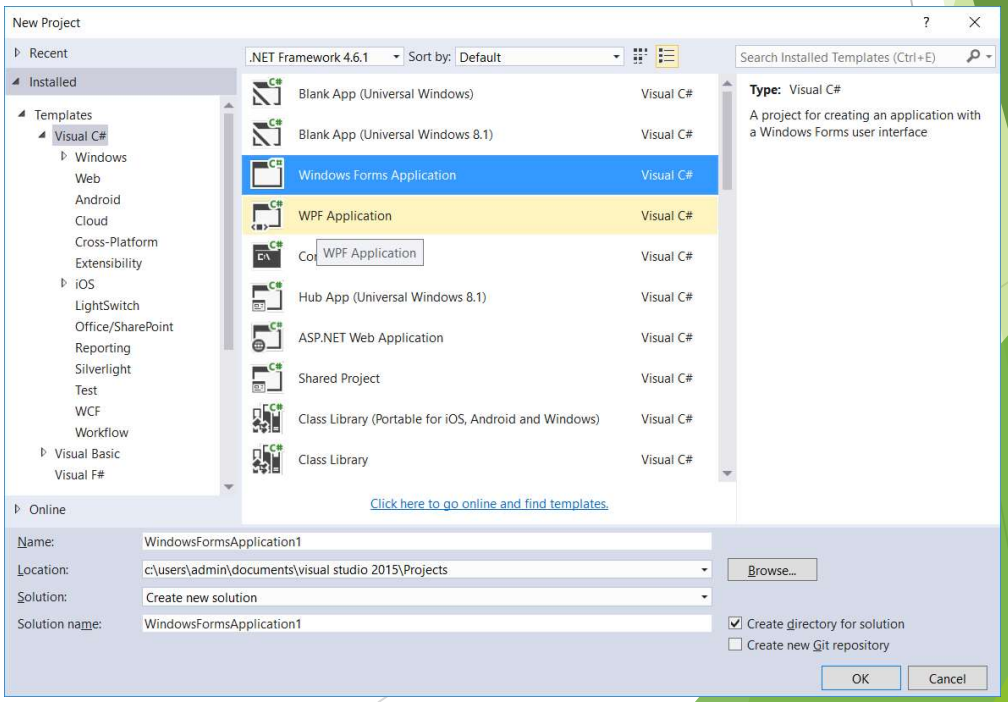
Čas 2.

Dodavanje projekta:

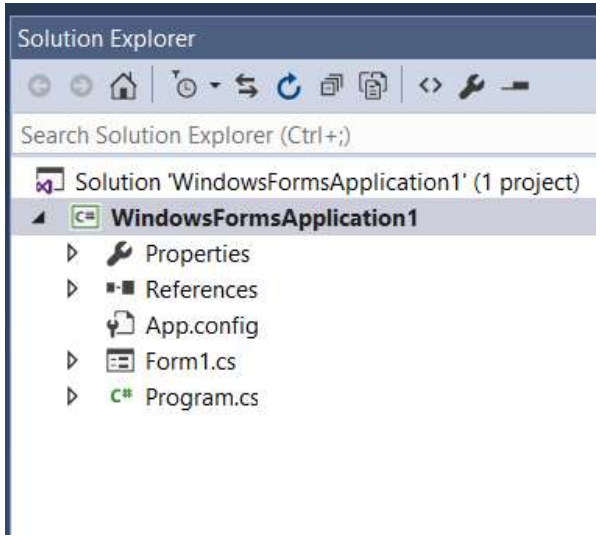
- Moguće je postojećem projektu dodati novi i tako imati u jednom radnom okruženju u okviru jednog rešenja više projekata.



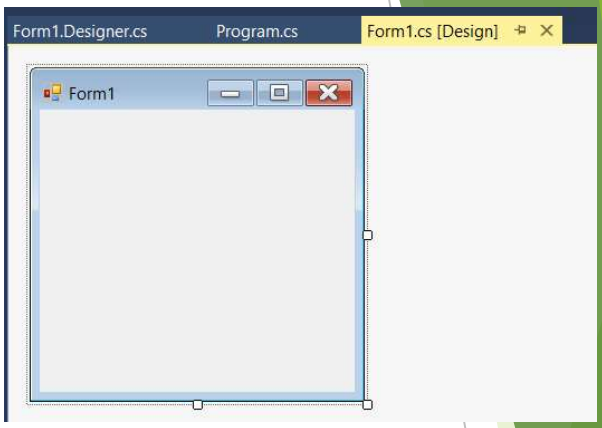
- Izbor tipa projekta.



- Dobijena struktura projekta:



- Izgled početne forme Form1 iz „Dizajnera“:



- Pokretanje aplikacije i startna forma Form1:

```
static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
}
```

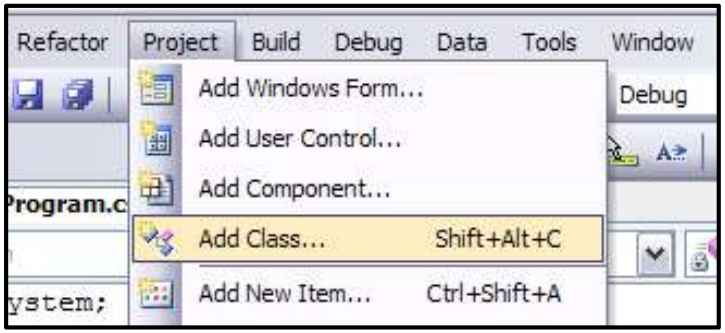
- Početni kod pripremljen za dalje programiranje:

```
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

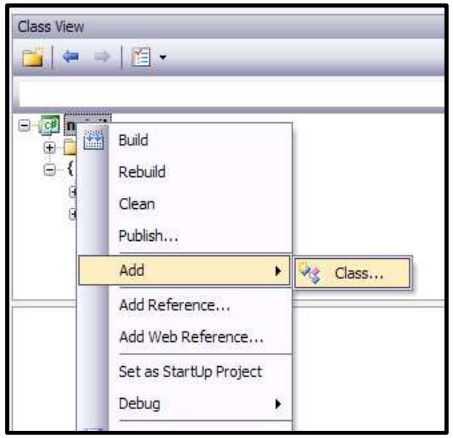
Metoda koja sadži kod „Dizajnera“ u drugom fajlu

Dodavanje nove klase projektu...

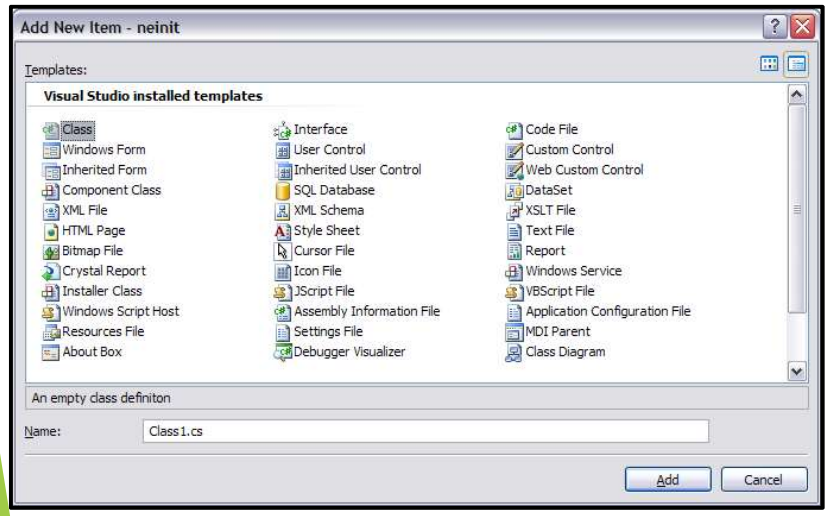
- 1. Iz glavnog menija:



- 2. Iz padajućeg menija Class View:

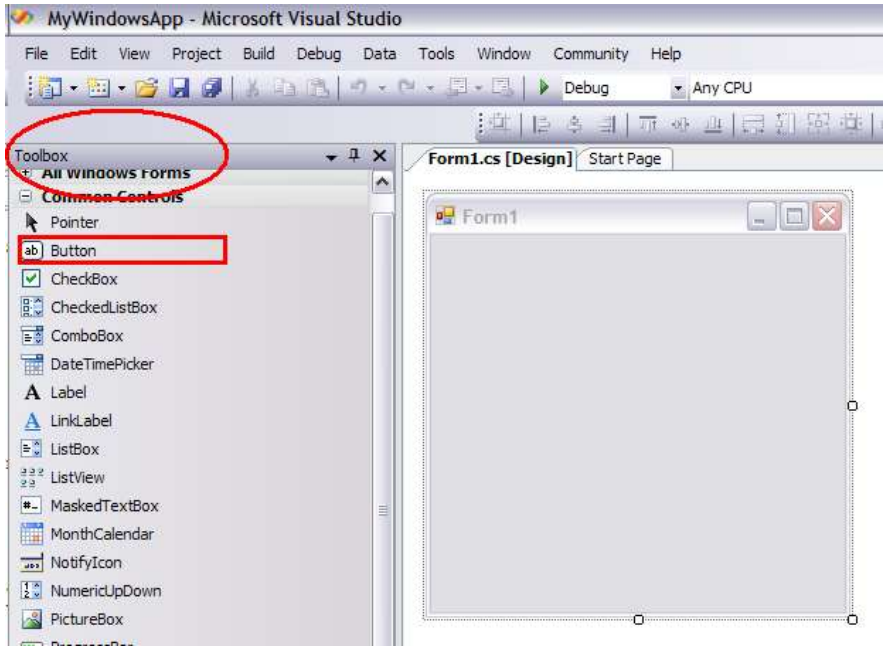


- 3. Preko stavke u meniju *Add New Item*:



- Podrazumevano pri dodavanju nove klase važi:
  - Nova klasa pripada istom prostoru imena koji je tekući!
  - Nova klasa se smešta u novi fajl sa istim imenom kao što je ime klase.

Dodavanje vizuelnih kontrola iz IDE okruženja



Podrazumevani događaj neke kontrole...

- ... se generiše dvosktrukim klikom na tu kontrolu. Za dugme to je klik mišem na dugme.
- *Ovo je dovoljno dok ne budemo radili događaje posebno*

Generisani kod!

Form1.cs | Form1.cs [Design] | Start Page

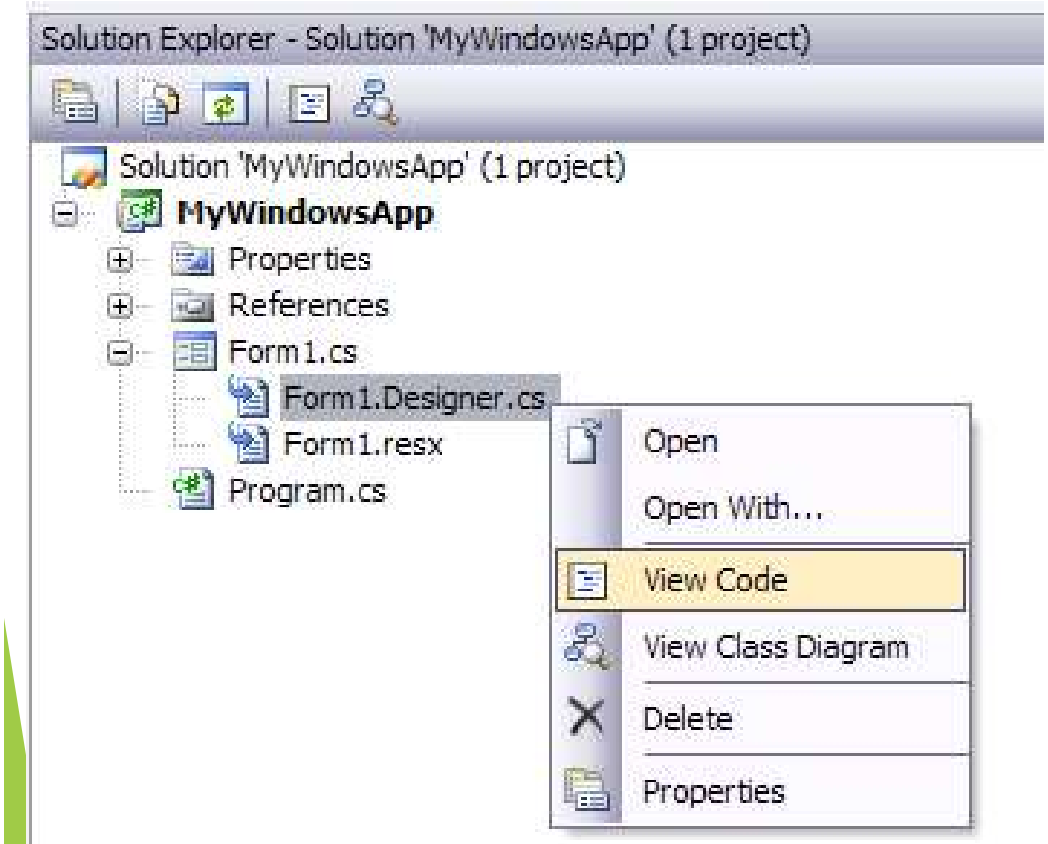
MyWindowsApp.Form1 | button1_Click(object sende

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace MyWindowsApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Ovo je odziv na klik misa");
        }
    }
}
```

Gde je dugme?

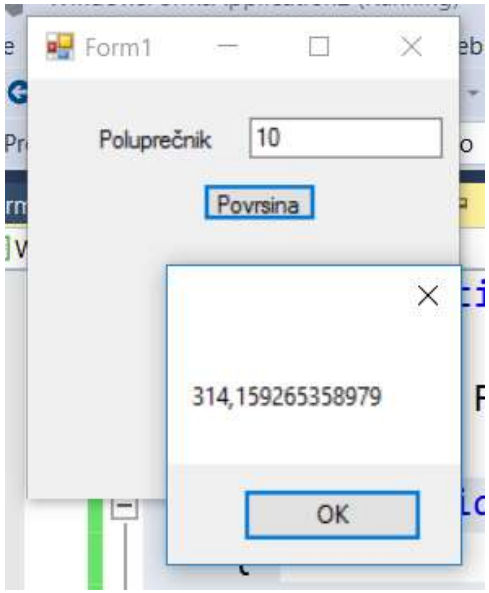
- Sve što uradite koristeći pomoće alate IDE okruženja rezultuje kodom.
- Neki delovi koda su sakriveni, ali ne i nedostupni.
- Namenjeni su prvenstveno za rad iz **dizajnera.**

Pogled na prozor “Solution Explorer”



Klasa u dva fajla!!

```
        base.Dispose(disposing);
    }

    Windows Form Designer generated code
}
#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.button1 = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(110, 84);
    this.button1.Name = "button1";
    this.button1.Size = new System.Drawing.Size(75, 23);
    this.button1.TabIndex = 0;
    this.button1.Text = "button1";
    this.button1.UseVisualStyleBackColor = true;
    this.button1.Click += new System.EventHandler(this.button1_Click);
    //
    // Form1
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
...
```



Nizovi u C#

- **REFERENTNOG** TIPA.
- Jednodimenzioni
- Višedimenzioni
  - Pravougaoni
  - Nazubljeni

```
static void Main() {

        int[] arr = new int[5]; // jednodimenzionalni niz

        for (int i = 0; i < arr.Length; i++)
        arr[i] = i * i;
        for (int i = 0; i < arr.Length; i++)
                Console.WriteLine("arr[{0}] = {1}", i,

}
```

```
arr[0] = 0
arr[1] = 1
arr[2] = 4
arr[3] = 9
arr[4] = 16
```

Jednodimenzioni niz referentnog tipa

```
static void Main() {
myClass[] arr = new myClass[5];
      for (int i = 0; i < arr.Length; i++)
      {
            arr[i] = new myClass(); // poz. konstruktor za svaki el.niza
      }
}
```

int[] arr = new int[5];

Zaključak:

- Potpuna analogija kod nizova referentnog odnosno vrednosnog tipa.
- Elemente niza referentnog tipa treba dodatno napraviti pozivom metode **new** za svaki elemenat niza.

Upotreba *foreach* petlje

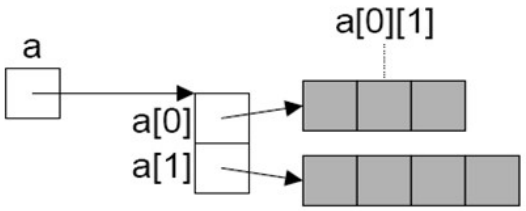
- Formiranje petlje koja će obuhvatiti rad sa svakim elementom niza je često u praksi. Ovo se lako izvodi foreach petljom. Pogledajte primer:

```
int[] n = new int[10] { 2,3,2,3,4,4,5,6,7,1};
int suma = 0;
for (int i = 0; i < 10; i++)
{
    suma += n[i];
}

suma = 0;
foreach (int el in n)
{
    suma += el;
}
```

Višedimenzionalni nizovi

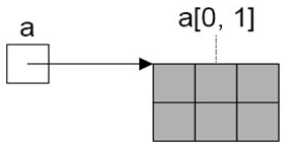
Primeri:

```
int[] a1;   // single-dimensional array of int
int[,] a2;  // 2-dimensional array of int
int[,,] a3; // 3-dimensional array of int
int[][] j2;  // niz nizova
int[][][] j3;// niz "niza nizova"
```




```
int[][] a = new int[2][];
a[0] = new int[3];
a[1] = new int[4];

int x = a[0][1];
int len = a.Length;  // 2
len = a[0].Length;   // 3
```

Nazubljeni niz (niz nizova)



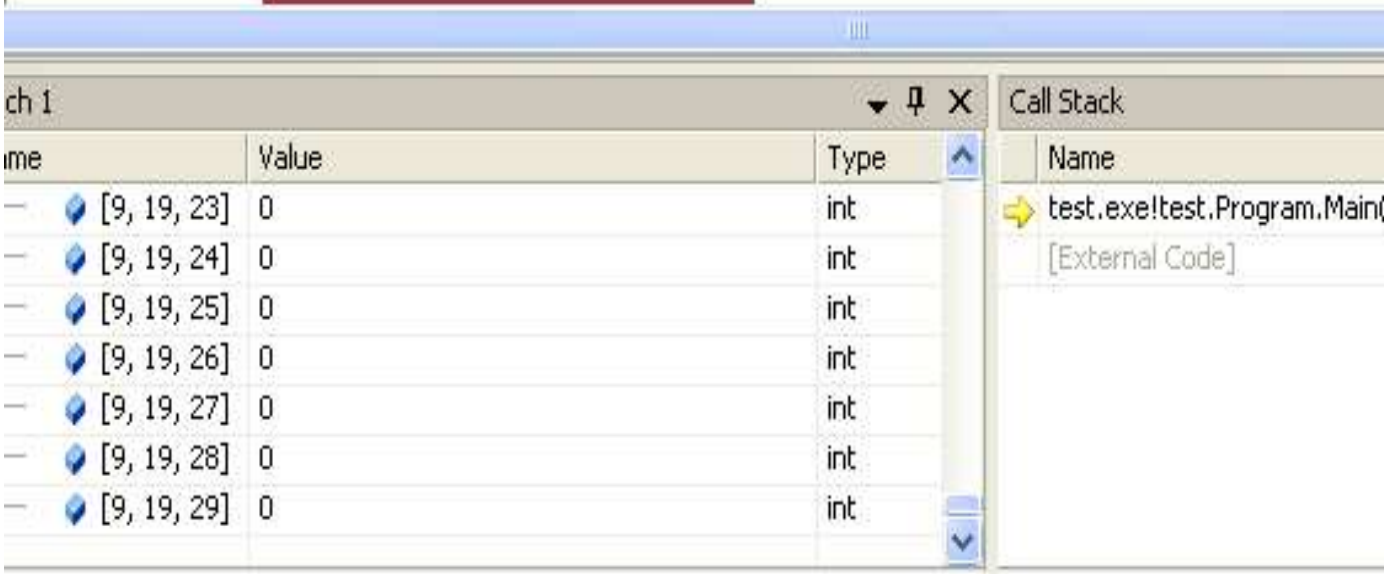

```
int[,] a = new int[2, 3];

int x = a[0, 1];
int len = a.Length;       // 6
len = a.GetLength(0); // 2
len = a.GetLength(1); // 3
```

Pravougaoni niz

Primeri:

- int[] a1 = new int[] {1, 2, 3};
- int[,] a2 = new int[,] {{1, 2, 3}, {4, 5, 6}};
- int[,,] a3 = new int[10, 20, 30]; //array type of int[,,]
- int[][] j2 = new int[3][];
  j2[0] = new int[] {1, 2, 3};
  j2[1] = new int[] {1, 2, 3, 4, 5,
  j2[2] = new int[] {1, 2, 3, 4, 5,

```
int[] a1 = new int[] { 1, 2, 3 };
int[,] a2 = new int[,] { { 1, 2, 3 }, { 4, 5, 6 } };
int[, ,] a3 = new int[10, 20, 30]; //array type of int[,,
int[][] j2 = new int[3][];
j2[0] = new int[] { 1, 2, 3 };
j2[1] = new int[] { 1, 2, 3, 4, 5, 6 };
j2[2] = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```



Višedimenzioni nizovi ref. tipa

- Potpuna analogija sa nizovima vrednosnog tipa sa obaveznonom alokacijom memorije i pozivom konstruktora što se obavlja pomoću operatora **new**

- Primeri:
- Kako metodi proslediti niz? Kako da metoda vrati kao rezultat neki niz?
- Napisati konzolnu aplikaciju za sortiranje niza vrednosti.

- Nizovi se koriste i za prosleđivanje više vrednosti nekoj metodi.
- C# ima ključnu reč ***params*** koja mogućnost da se ovakvi parametri prosleđuju kao promenljivi broj argumenata.
- Ovaj niz mora biti jednodimenzioni.

Strukture u C#

```
struct Point
{
    public int x;
    public int y;
    public Point(int x1, int y1)
    {
        x = x1;
        y = y1;
    }
}
```

```
Point p1;  // p1.x, p1.y su nule
Point p2 = new Point(2,2); // p2.x, p2.y
```

- 1. VREDNOSNI TIP
- Kako se piše jedna struktura u odnosu na klasu?
  - Velika sličnost.
  - Umseto ključne reči class sledi struct
  - Struktura može da sadrži polja i metode kao i klasa.
- Prilikom deklaracije se ujedno formira i njena instanca jer je vrednosnog tipa. Pri kreiranju objekta tj. Instance nije neophodno da se koristi operator ***new***, ali nije ni zabranjeno upotrebiti ga. Pozivom ovog operatora definiše se metoda za inicijalizaciju vrednosti tj. određuje se poziv konstruktora. *(Instanca strukture se formira na steku i ne može se preusmeriti na dinamičku memoriju upotrebom new operatora.)*

Strukture u C#: Primer

```
struct Point
{
        public int x, y;
        Point (Point p) { x = p.x; y = p.y; }
        void MoveTo (int a, int b) { x = a; y = b; }
}
Point p = new Point(3, 4);
Point q;
p.MoveTo(10, 20);// method call
q.MoveTo(60, 20);
```

Point point; // IDENTIČNO SA
Point point = new Point();

Strukture u C#

- Izvedene su iz klase Object (kao int), ali ne mogu se koristiti kao roditeljske klase za druge strukture odnosno klase. Znači, struktura ne sme da nasleđuje druge strukture ili klase.

- Nasleđivanje interfejsa je dozvoljeno ali sa obavezom da se realizuju deklarisane metode u interfejsu. (Kasnije će biti rađeni interfejsi)

- Dodatna literatura: C# kroz praktične primere, Charles Wright, The McGraw-Hill Companiews, Osborne.

Upotreba ključne reči *this*

- Označava korišćenje objekta iste klase u kojoj se koristi.
- Najčešće ga koristi jer inicira pomoć u pisanju koda i nije neophodan.
- Na primer: U prethodnom primeru se može napisati konsturuktor na sledeći način:
  - Point (Point p) { this.x = p.x; this.y = p.y; }
- Nekada je neophodan. Ako se piše metoda čiji argument objekat koji se koristi.

Rad sa stringom: Klasa
System.String - string

- Povezivanje više stringova operatorom **+**: "Don " + s
- Indeksiranjem dobija se vrednost karaktera na poziciji: s**[**i**]**
- Svojsvto **Length** daje dužinu stringa: s.Length
- **Referencni tip podataka**
  - Ali zbog specifičnosti tipa, usaglašavanja sa C sintaksom i pozitivnih iskustava, napravljeni su **izuzeci.**

1. Nije neophodno pozivanje eksplicitnog konstrukrora pri inicijalizaciji stringa: ***string s = "perica";***
2. Vrednosti se mogu porediti neposredno: ***s=="perica"***
   - (dva stringa su jednaka, ==, ako imaju iste karaktere na svim pozicijama ili su null oba)

- Klasa String definiše mnoge korisne operacije nad stringovima (metode)
  - *CompareTo, IndexOf, StartsWith, Substring, Replace...*

Osobenosti strukture

Klasa	Struktura
Referencni tip (objekti tj instance klase se skladište u din. mem. tj. heap-u)	Vrednosni tip (objekti se čuvaju na steku)
Može da nasleđuje (sve klase su izvedene iz klase object)	Ne može da nasleđuje (ali kompatibilna sa klasom object)
Mogu implementirati interfejse	Mogu implementirati interfejse
Mogu imati destruktor	Ne mogu imati destruktor

Dostupnost tj. mogućnost pristupa

- **public** – Pristup bez ograničenja.
- **protected** – Moguć pristup u istoj klasi i klasama izvedenim od te klase.
- **internal** – Pristup ograničen na tekući program.
- **protected internal** – Pristup ograničen na program ili na tipove izvedene iz te klase.
- **private** – Moguć pristup samo u istoj klasi.

Nabrojive liste

- **Enumeratori ili nabrojive liste**
  - Vrednosni tip podataka
  - Sintaksa:

    enum *NAZIV*

    { vrednost1, vrednost2, … vrednostN };

```
public enum Dani { Ponedelja, Utorak, Sreda,
Dani dan;
public void test()
{
    dan = Dani.
}

public const                = 12;
public int M
public stati                icField = 34;

public void MyMethod()
```

- Četvrtak
- Nedelja
- Petak
- Ponedelja
- Sreda
- Subota
- Utorak

Primer 2.

```
enum Color {red, blue, green} //vrednosti 0, 1, 2
enum Access {personal=1, group=2, all=3}
enum Access1 : byte {personal=1, group=2, all=3}

Color c = Color.blue;
a = Access.personal | Access.group; // !!!
if ((Access.personal & a) != 0) Console.WriteLine("access
   granted");
```

Operacije na nabrojivim listama

- Poređenje:
  - if (c **==** Color.red) ...
  - if (c **>** Color.red && c **<=** Color.green) ...
- Operatori: +, -, ++, --, &, |, ~
  - c = c **+** 2;
  - c**++**;
  - if ((c **&** Color.red) == 0) ...
  - c = c **|** Color.blue;
  - c = **~** Color.red;

- Kompajler ne proverava da li rezultat ima validnu vrednost.
- Enumeratori ne mogu biti pridruženi celobrojnim vrednostima osim ako postoji eksplicitno kastovanje.
- Enumeratori nasledjuju *object*, sledi da poseduju: Equals, ToString, ...
- Klasa **System.Enum** omogućuje operacije nad ovim podacima kao na primer (GetName, Format, GetValues, ...).

Primer 3.

```
enum Temperatures
  {
     WickedCold = 0,
     FreezingPoint = 32,
     LightJacketWeather = 60,
     SwimmingWeather = 72,
     BoilingPoint = 212,
  }
  static void Main()
  {
     System.Console.WriteLine( "Freezing point of water: {0}",
     ( int ) Temperatures.FreezingPoint );
     System.Console.WriteLine( "Boiling point of water: {0}",
     ( int ) Temperatures.BoilingPoint );
  }
```

Generički tipovi

- Uvedeni u .Net 2.0
- Generički tipovi se generišu pomoću drugog tipa. Ovi tipovi se prepoznaju jer pri generisanju mora da bude naveden tip za koji se generišu.
- Po prirodi predstavljaju šablone za generisanje realni tipova.
- Pogledajmo jednu generičku metode:

```
void zamena<T>(ref T a, ref T b)
{
    T tmp;
    tmp = a;
    a = b;
    b = tmp;
}
```

- Primer generičke strukture:

```
struct GenStruct1<T>
{
    public T var1;
    public string add(T t)
    {
        return "" + var1 + t;
    }
}
```

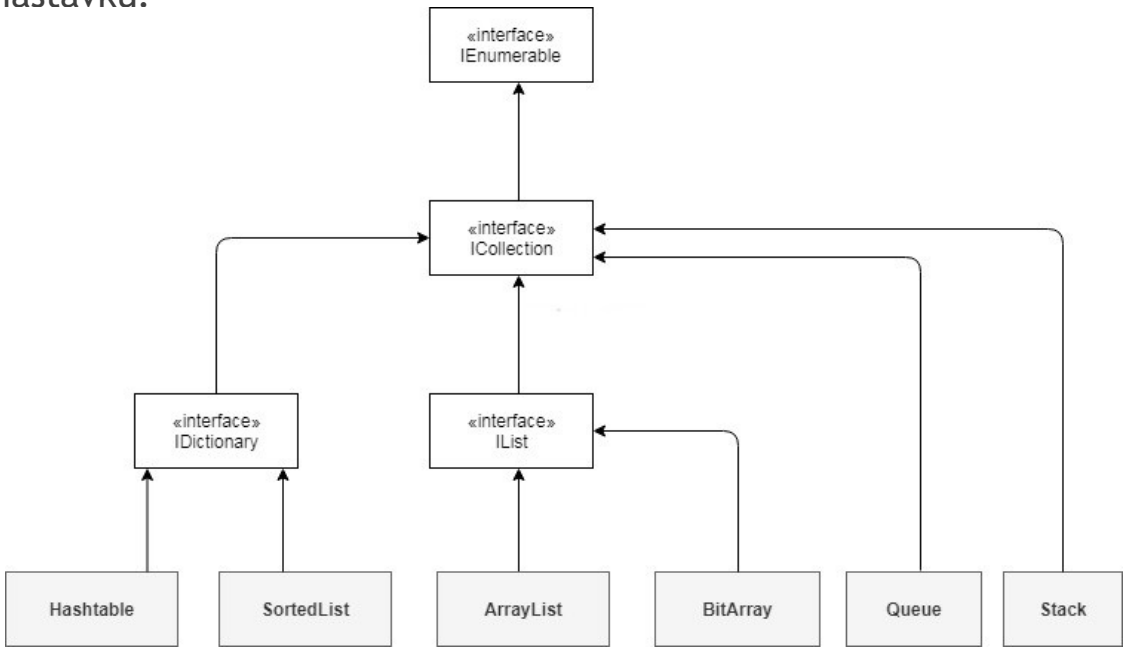
- Upotreba generičke strukture:

```
GenStruct1<int> a1;
GenStruct1<string> a2;
a1.var1 = 3;
a2.var1 = "tri";

Trace.WriteLine(a1.add(4));
Trace.WriteLine(a2.add("četiri"));
```

- Napomena: Zapazite da se rezultat prikazuje u pomoćnom Output prozoru.
  Ovo se postiže primenom klase Trace iz imenskog prostora
- `System.Diagnostics`

Kolekcije

- Postoje generičke i negeneričke kolekcije
- Prikaz hijerarhije klasa kolekcija dat je na slici. Detaljiji opis sledi u nastavku.



Napomena:

- Klasa System.Array koja je bazna klasa svih tivoa u C#, takođe implementira generički interfejs *IList*. Implementacija metoda ipak se izvodi na poseban način ostavljajući razliku u sintaksi.

IEnumerable

Metode za interface IEnumerator<T>

- T Current
- bool MoveNext()
- void Reset()

ICollection

Sopstvene metode za interfejs ICollection<T>

- bool Contains(T element)
- void Add(T element)
- bool Remove(T element)
- void Clear()
- void CopyTo(T[] targetArray, int startIndex)
- int Count

- bool IsReadOnly

IList

Sopstvene metode za interface IList<T>

- T this[int index]
- int IndexOf(T element)
- void Insert(int index, T element)
- void RemoveAt(int index)

IDictionary

Sopstvene metode za interface IDictionary<K,V>:

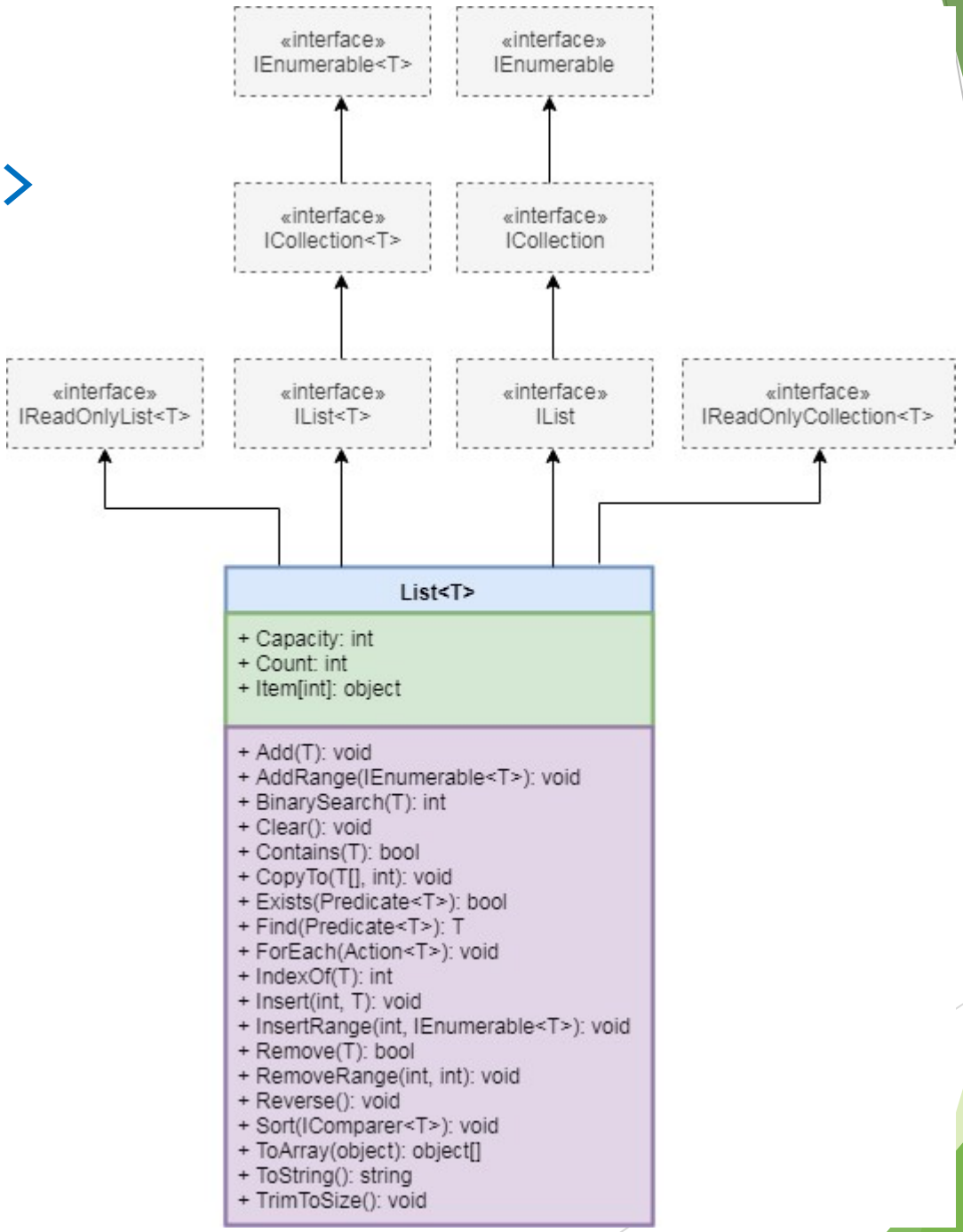
- Nasleđuje: ICollection<KeyValuePair<K, V>>
- Nasleđuje: IEnumerable<KeyValuePair<K, V>>
- V this[K key] - poseduje tzv.getter i setter;
- void Add(K key, V value) – ako key već nije dodato
- bool Remove(K key)
- bool ContainsKey(K key)
- bool TryGetValue(K key, out V value)
- ICollection<K> Keys - getter
- ICollection<V> Values - getter

Generičke kolekcije

- Prostor imena: ***System.Collections.Generic***
- Zapažamo podgrupu koja radi sa parovima.

List<T>	Sadrži listu određenog tipa. Automatski raste sa dodavanjem el.
Hashset<T>	Sadrži neduplirane elemente.
Queue<T>	Čuva vrednosti na FIFO način. Metodom Enqueue() se dodaje vrednost a metodom Dequeue() izbacuje iz kolekcije.
Stack<T>	Čuva vrednosti na LIFO način. Metodom Push() se dodaje a metodama Pop() odnosno Peek() se uzimaju vrednosti.
Dictionary<K,T>	Sadrži parove key-value.
SortedList<K,T>	Lista parova koji se automatizovano sortiraju.
SortedDictionary<K,T>	Lista parova koji se automatizovano sortiraju.

Liste
List<T>


«interface»
IEnumerable<T>
«interface»
IEnumerable
«interface»
ICollection<T>
«interface»
ICollection
«interface»
IReadOnlyList<T>
«interface»
IList<T>
«interface»
IList
«interface»
IReadOnlyCollection<T>
List<T>
+ Capacity: int
+ Count: int
+ Item[int]: object
+ Add(T): void
+ AddRange(IEnumerable<T>): void
+ BinarySearch(T): int
+ Clear(): void
+ Contains(T): bool
+ CopyTo(T[], int): void
+ Exists(Predicate<T>): bool
+ Find(Predicate<T>): T
+ ForEach(Action<T>): void
+ IndexOf(T): int
+ Insert(int, T): void
+ InsertRange(int, IEnumerable<T>): void
+ Remove(T): bool
+ RemoveRange(int, int): void
+ Reverse(): void
+ Sort(IComparer<T>): void
+ ToArray(object): object[]
+ ToString(): string
+ TrimToSize(): void

Liste

- Lista je skup objekata, istog tipa - generička lista, bez unapred poznatog broja, ondnosno sa mogućnošću da se naknadno u već formiranu listu dodaju ili izbacuju elementi.
- Ako se radi o generičkoj listi:
- `List<tip> t = new List<tip>();`
- Dodavanje elementa listi:
- `t.Add(22); t.Add(123);`
- Brisanje cele listi:
- `t.Clear();`
- Izbacivanje elementa iz liste
- `t.Remove(obj);` ili
- `t.RemoveAt(2);`

- Konstruktori: `List(), List(IEnumerable<T>), List(int)`
- Inicijalizacija kolekcije: `new List<T> { t1, t2, ..., tn }`
- Pristup: `this[int], GetRange(int, int)`
- Mere: `Count, Capacity`
- Dodavanje elemenata: `Add(T), AddRange(IEnumerable<T>), Insert(int, T),InsertRange(int, IEnumerable<T>)`
- Izbacivanje elemenata: `Remove(T), RemoveAll(Predicate<T>), RemoveAt(int), RemoveRange(int,int), Clear()`
- Reorganizacija: `Reverse(), Reverse(int, int), Sort(), Sort(Comparison<T>),Sort(IComparer<T>), Sort(int, int, IComparer<T>)`
- Pretraga: `BinarySearch(T), BinarySearch(int, int, T, IComparer<T>), BinarySearch(T,IComparer<T>), Find(Predicate<T>), FindAll(Predicate<T>), FindIndex(Predicate<T>), FindLast(Predicate<T>), FindLastIndex(Predicate<T>), IndexOf(T), LastIndexOf(T)`
- Upiti true/false: `Contains(T), Exists(Predicate<T>), TrueForAll(Predicate<T>)`

- Konverzije: `ConvertAll<TOutput>(Converter<T, TOutput>), CopyTo(T[]),`

Skupovi - ISet

- Generički interfejs za realizaciju skupova. Neki primeri kalsa: `HashSet`, `SortedSet`, `ImmutableHashSet`, `ImmutableSortedSet`.
- Specifične metode se odnose na rad sa skupovima:
  - `IntersectWith(IEnumerable<T>)`
  - `Overlaps(IEnumerable<T>)`
  - `UnionWith(IEnumerable<T>)`
  - `IsProperSubsetOf(IEnumerable<T>)`
  - `. . .`

- Primer primene skupva:

```
string[] nizVocki = {"visnja", "jabuka", "sljiva",
                     "jabuka", "kruska",  "visnja"};

// Prikaz niza nizVocki
Console.WriteLine(string.Join(",", nizVocki));

// Kreiranje skupa
var vrsteVoca = new HashSet<string>(nizVocki);

// Prikaz niza vrsteVoca.
Console.WriteLine(string.Join(",", vrsteVoca.ToArray()));
```

Rečnik – Dictionary<K,V>

- Ova kolekcija odgovara realnom rečniku pojmova, s tom razlikom što se ovde radi o objektima. Dake, to je kolekcija ključnih objekata i pratećih objekata. Kaže se da radi sa parovima ključ, vrednost tj. K,V.
- Novi element rečnika mora imati ključ koji ne odgovara ni jednom postojećem elementu.

```
IDictionary<int, string> redni = new Dictionary<int, string>();
// dodavanje
redni.Add(1, "Prvi");
redni.Add(2, "Drugi");
redni.Add(3, "Treći");
// svojstva..
int brEl = redni.Count(); //brEl = 3
bool samoCitanje = redni.IsReadOnly; // false
// provera kljuca
bool postoji_kljuc_3 = redni.ContainsKey(3); // true
// greška
redni.Add(3, "Treci"); // greska
```

Hashtable

- Hashtable je negenerički tip.
- Nije tipiziran tako da ključevi i vrednosti mogu biti bilo kog tipa.
- Vrednosti zahtevaju tzv. *boxing*/*unboxing*. Kasnije više…
- Kada se pokuša pristp po ključu koji ne postoji Hashtable vraća null.
- Nema uređivanja redosleda.
- Sporiji zbog *boxing*/*unboxing* operacija.

- Primer:

```
Hashtable ht = new Hashtable();
ht.Add(1, 3.14159);
ht.Add("002", "C#");
ht.Add(5.5, "ASP.Net");

ICollection keys = ht.Keys;

foreach (Object k in keys)
{
    Console.WriteLine("Kljuc:{0} Vrednost{1}", k, ht[k]);

}
```

Sortirana lista - SortedList

- Može se koristit kao generička i negenerička.
- Generička SortedList predstavlja kolekciju parova ključ/vrednost koji su sortirani po ključu (na osnovu pridruženog IComparer <T>). SortedList kolekcija čuva parove ključeva i vrednosti u rastućem redosledu ključa.

```
SortedList<int, string> redni = new SortedList<int, string>();
// dodavanje
redni.Add(3, "Treći");
redni.Add(0, "Nulti");
redni.Add(1, "Prvi");
redni.Add(2, "Drugi");

string st2 = redni[2]; // Drugi
string st1 = redni[1]; // Prvi
```

Red - Queue

- Može biti generički i negenerički.
- Predstavlja kolekciju tipa ”First in First Out” - (FIFO).
- Osim svojstva **Count** koji predstavlja broj elemenata u kolekciji postoje I sledeće metode:

public virtual void Clear(); brišu se svi elementi u redu.
public virtual bool Contains(object obj); - određuje da li je element u redu
public virtual object Dequeue(); - izbacuje prvi objekat iz reda i isti je povratna vrednost metode.
public virtual void Enqueue(object obj); - dodaje objekat na kraj reda.
public virtual object[] ToArray(); - kopira elemente reda u niz.

- Primer:

```
Queue qq = new Queue();
qq.Enqueue(1);
qq.Enqueue(2);
qq.Enqueue("treci");

int prvi_1 = (int)qq.Peek(); // 1
int prvi_2 = (int)qq.Peek(); // 1


int x1 = (int)qq.Dequeue(); // 1
int x2 = (int)qq.Dequeue(); // 2
string x = (string)qq.Dequeue(); // "treci"
qq.Dequeue(); // greska
```

Stek podataka - Stack

- Može biti generički i negenerički tip.
- Predstavlja klasu za rad sa kolekcijom koja poseduje LIFO mehanizam čuvanja elemenata. To znači posledne dodat element je element koji se prvi dohvata sa steka.
- Osim svojstva Count postoje sledeće bitne metode:

public virtual void Clear(); - brišu se svi elementi na steku.
public virtual bool Contains(object obj); - određuje da li je element na steku.
public virtual object Peek(); - vraća objekat koji je na vrhu steka.
public virtual object Pop(); - izbacuje i vraća objekat na vrhu steka.
public virtual void Push(object obj); - ubacuje objekat na vrh steka.
public virtual object[] ToArray(); - vraća niz podataka Stack.

- Primer

```
Stack ss = new Stack();
ss.Push(1);
ss.Push(2);
ss.Push("treci");

string x1 = (string)ss.Peek(); // treci
string x2 = (string)ss.Peek(); // treci

string x = (string)ss.Pop(); // "treci"
int a2 = (int)ss.Pop(); // 2
int a1 = (int)ss.Pop(); // 1
ss.Pop(); // greska
```

Klasa Array

- **(Apstraktna)** klasa Array nije deo Sistem.Collections imenskog prostora. Smatra se kolekcijom jer implementira interfejs IList.
- Array je bazna klasa za jezičku realizaciju nizova. Implementira: ICollection IEnumerable IList IStructuralComparable IStructuralEquatable ICloneable
- Element je vrednost u nizu. Dužina polja je ukupan broj elemenata koje može da sadrži. Donja granica polja je indeks njegovog prvog elementa. Array može imati bilo koju donju granicu, ali podrazumevano ima donju granicu nule. Različita donja granica može biti definisana prilikom kreiranja instance klase Array koristeći CreateInstance.A višedimenzionalni niz može imati različite granice za svaku dimenziju. Niz može imati najviše 32 dimenzije.
- Za razliku od klasa u imenskom prostoru Sistem.Collections, Array ima fiksni kapacitet. Da biste povecali kapacitet, morate kreirati novi Array objekt sa potrebnim kapacitetom, kopirati elemente iz starog Array objekta u novi, i izbrisati stari Array.
- Podrazumevano, maksimalna veličina polja je 2 gigabajta (GB). U 64-bitnom okruženju, možete izbeci ograničenje veličine tako što cete podesiti atribut omogucen za konfiguracioni element gcAllowVeryLargeObjects u *true* u okruženju vremena izvršavanja. Međutim, niz ce i dalje biti ograničen na ukupno 4 milijarde elemenata, i na maksimalni indeks 0Ks7FEFFFFF u bilo kojoj datoj dimenziji (0Ks7FFFFFC7 za bajtne nizove i nizove jednobajtnih struktura).

```
Array myArr = Array.CreateInstance(typeof(Int32), 2, 3, 4);
for (int i = myArr.GetLowerBound(0); i <= myArr.GetUpperBound(0); i++)
{
    for (int j = myArr.GetLowerBound(1); j <= myArr.GetUpperBound(1); j++)
    {
        for (int k = myArr.GetLowerBound(2); k <= myArr.GetUpperBound(2); k++)
        {
            myArr.SetValue((i * 100) + (j * 10) + k, i, j, k);
            int test = (int)myArr.GetValue(i, j, k);
        }
```

U C # 2.0 i kasnije, jednodimenzionalni nizovi koji imaju donju granicu 0 automatski implementiraju IList <T>.
Ovo omogućava da se kreiraju generičke metode koje mogu koristiti isti kod za iteraciju kroz nizove i druge tipove kolekcija. Ova tehnika je primarno korisna za čitanje podataka u kolekcijama.
IList <T> interfejs se ne može koristiti za dodavanje ili uklanjanje elemenata iz niza. Izuzetak će biti izbačen ako pokušate da pozovete IList <T> metodu kao što je RemoveAt na nizu u ovom kontekstu.

```
static void Main()
{
    int[] arr = { 2,4,6 };
    List<int> list = new List<int>();

    foreach (int x in arr)
        list.Add(x);

    Console.WriteLine("**************Test niza**************");
    testKolekcije<int>(arr);
    Console.WriteLine("**************Test liste**************");
    testKolekcije<int>(list);
}
```

```
static void testKolekcije<T>(IList<T> nn)
{
    System.Console.WriteLine("IsReadOnly = {0}", nn.IsReadOnly);
    try
    {
        nn.RemoveAt(1);
    }
    catch (Exception ex)
    {
        Console.WriteLine("greska " + ex.Message);
    }
    foreach (T item in nn)
        System.Console.Write(item.ToString() + " ");
}
```

ArrayList

- Predstavlja uređenu kolekciju objekta koji se može pojedinačno indeksirati. U osnovi je alternativa nizu. Međutim, za razliku od niza, možete dodati i ukloniti stavke sa liste na određenoj poziciji koristeći indeks i polje se automatski promeni. Takođe omogućava dinamičku dodelu memorije, dodavanje, pretraživanje i sortiranje stavki na listi.
- Pogledajmo svojstva i metode koje karakterišu ovu klasu.

- Primer:

```
ArrayList al = new ArrayList();

// formiranje arraylist-e
al.Add(44); al.Add(22); al.Add(11);
al.Add(121); al.Add(244); al.Add(33);

// svojstva
Console.WriteLine("Capacity: {0} ", al.Capacity);
Console.WriteLine("Count: {0}", al.Count);
al.Sort();
foreach (int a in al)
    Console.Write(a + " ");

al.Add("prvi");
al.Add("drugi");

foreach (Object a in al)
    Console.Write(a + " ");
```

Array vs ArrayList vs List

- To su različiti tipovi objekata. Oni imaju različite sposobnosti i čuvaju svoje podatke na različite načine.

- **Array** (Sistem.Arrai) je fiksan u veličini kada je dodeljen. Ne možete dodavati stavke ili ukloniti stavke iz njega. Takođe, svi elementi moraju biti istog tipa. Kao rezultat toga, on je bezbedan za tipove, a ujedno je i najefikasniji od tri, kako u pogledu memorije tako i performansi. Takođe, Sistem.Array podržava višestruke dimenzije (tj. Ima svojstvo Rank) dok List i ArraiList nemaju (iako možete da napravite listu lista ili listu ArrayListova, ako želite).

- **ArrayList** je fleksibilan niz koji sadrži listu objekata. Možete dodati i ukloniti stavke iz njega i automatski se bavi dodeljivanjem prostora. Ako u nju čuvate tipove vrednosti, oni su uokvireni i unboxed, što može biti malo neefikasno. Takođe, on nije bezbedan za tipove.

- **Lista <>** koristi generički tip; to je u suštini verzija ArraiList-a sigurna od tipa. To znači da ne postoji box ili unboxing (što poboljšava performanse) i ako pokušate da dodate stavku pogrešnog tipa, generisaće grešku vremena kompajliranja.

BitArray

- Klasa BitArray namenjena je radu sa bit vresnostima. Ove vrednosti se prikazuju kao tipvi Bool, kada je bit vrednosti 1 to se označava sa true i obrunuto.
- Koristi se kada se ne zna broj bita sa kojim se radi unapred.
- Specifične metode za rad sa bitima su:
- **And**
- **Not**
- **Or**
- **Set(int index, bool value)** – postavlja vrednost na određenu poziciju

- Primer:

```
BitArray ulaz = new BitArray(16);
BitArray core = new BitArray(8);
byte[] a = { 66, 22 };
byte[] b = { 7, 7 };

ulaz = new BitArray(a);
core = new BitArray(b);

Console.WriteLine("ulaz: 66,22");
for (int i = 0; i < ulaz.Count; i++)
    Console.Write("{0} ", ulaz[i]);

Console.WriteLine("\ncore: 7");
for (int i = 0; i < core.Count; i++)
    Console.Write("{0} ", core[i]);

Console.WriteLine("\n\nrez: ulaz AND core");
BitArray rez = new BitArray(16);
rez = ulaz.And(core);
```