QASJA NË TË DHËNAT E KOMPONETEVE DHE OBJEKTEVE

KAPITULLI 4

Prof. Ass. Dr. Isak Shabani

Qasja në të dhëna

- Të dhënave ne klasa, objekte dhe komponente mund të ju qasemi në disa mënyra.
- Qasja në të dhëna direkt nga variablat publike të instancës
  - Mos e bëni këtë!
- Qasja përmes ADO dhe ADO.NET
  - Lidhaj me bazat e te dhenave
- Qaja në mënyrë indirekte përmes vetive
  - Klientët nuk mund të vërejnë dallimin në mes të vetive dhe variablave publike të instancës
- Në mënyrë indirekte përmes metodave
  - Duhet rezervuar për "Ilogaritjet në object"
- Qaja indirekt përmes indeksimit
  - Ofron qasje në kuptimin e notacionit të njohur nga indeksimi tradicional i vargjeve
- Indirekt përmes operatorëve të mbingarkuar
  - Ofron qasje në kuptim të simboleve operacionale të definuara në një gjuhë

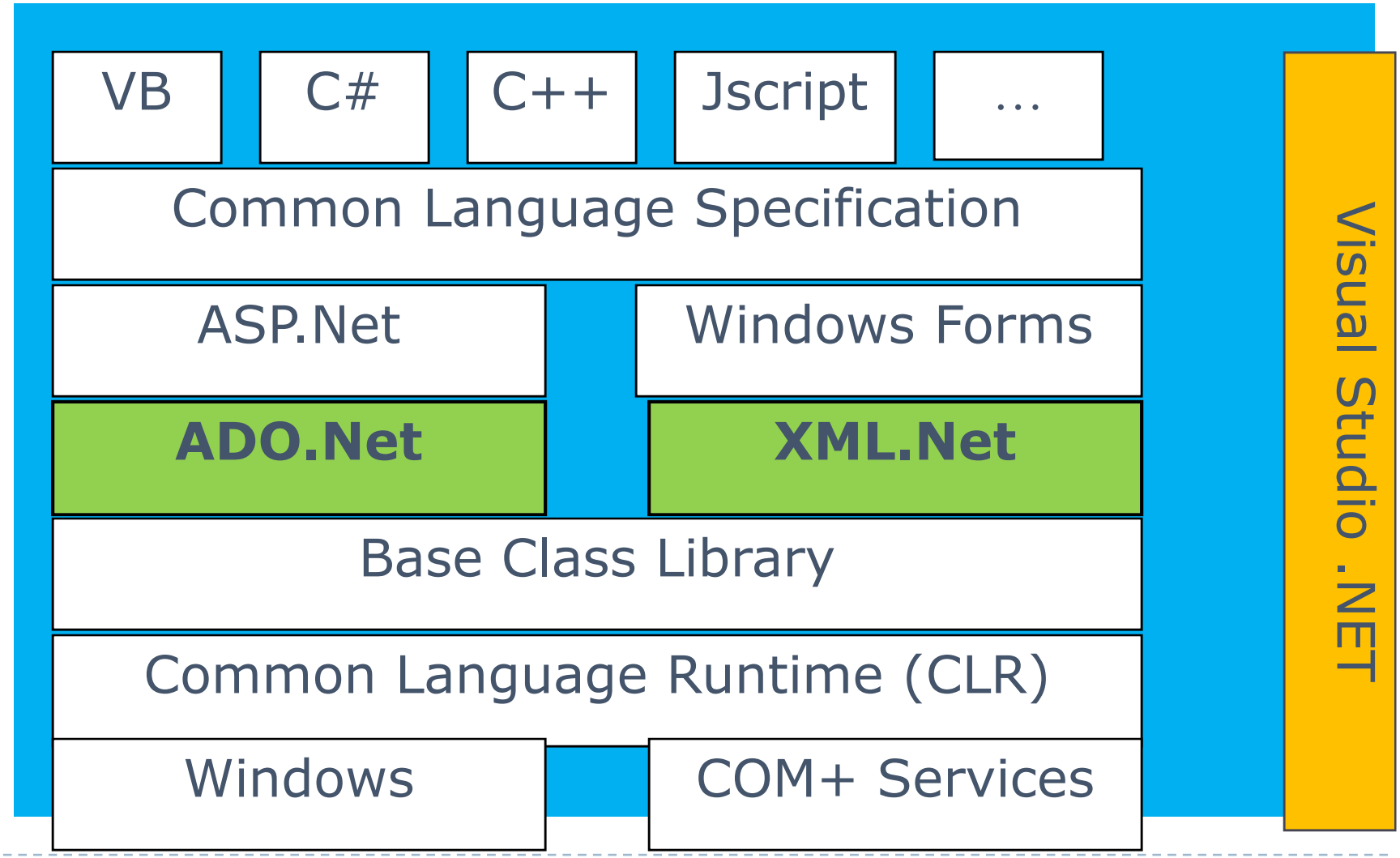
Hyrje ne komponeten ADO.Net

- Çfarë është ADO.Net ?
- Çfarë ndodh me ADO?
- Struktura objekt ADO.Net
- Lidhja
- Komandat
- Lexuesit dhe
- Grupet etë dhënave

Hyrje ne komponeten ADO.Net

- Klasat per qasje ne të dhëna ne kornzen .Net
- Dizajnimi me efikas per qasje ne të dhënat
- Mbështetje nga XML per kyçje dhe shkyçur ne rekordet e te dhenave

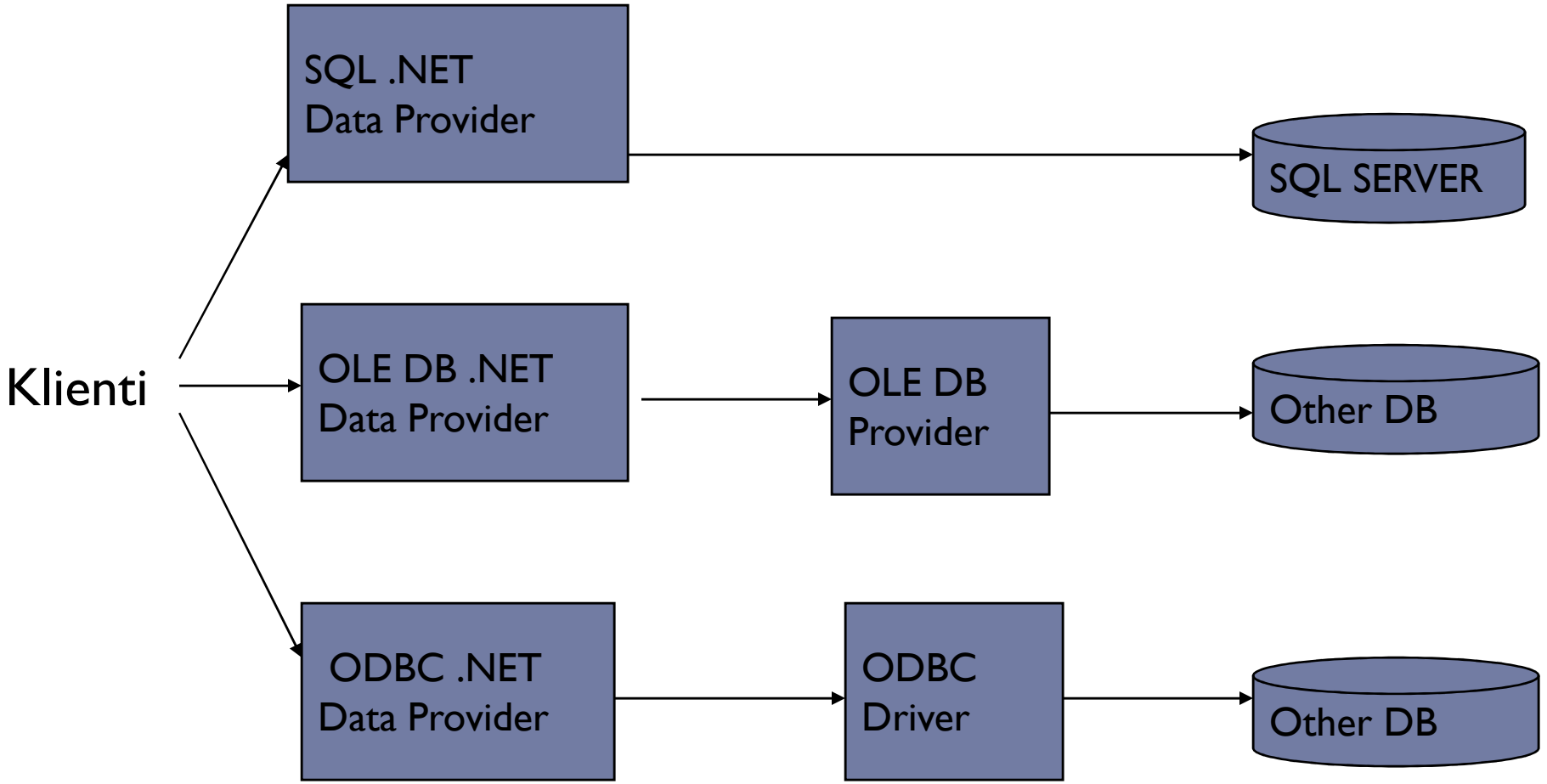
Komponenta ADO



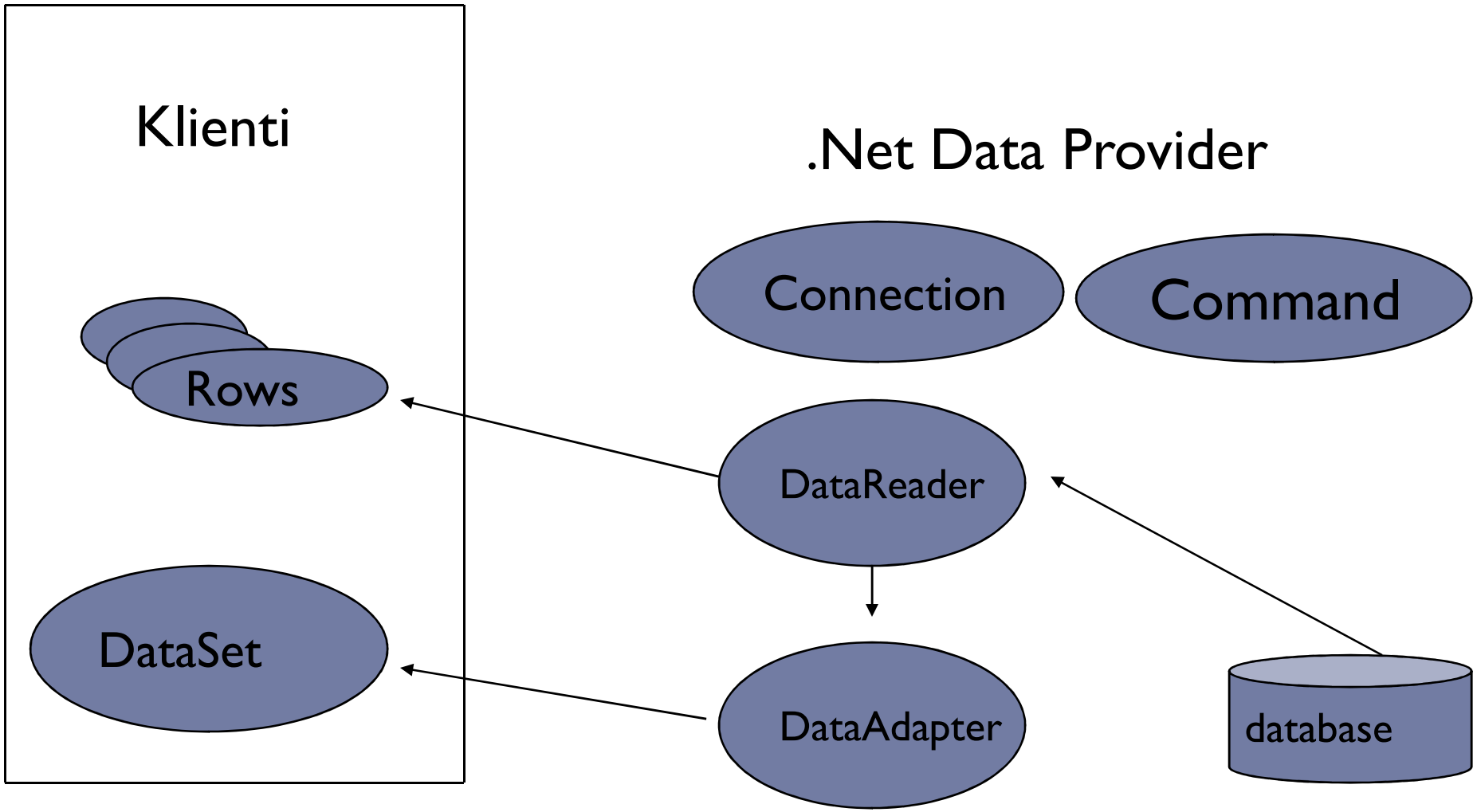
Krahsimi i ADO / ADO.Net

Tiparte	ADO	ADO.Net
të dhënat e shkëputur	Limited support, suitable for R/O	Strong support, with updating
Kalimi datasets	COM marshalling	DataSet support for XML passing
Shlallzueshmeria (Scalability)	Limited	Disconnected access provides scalability

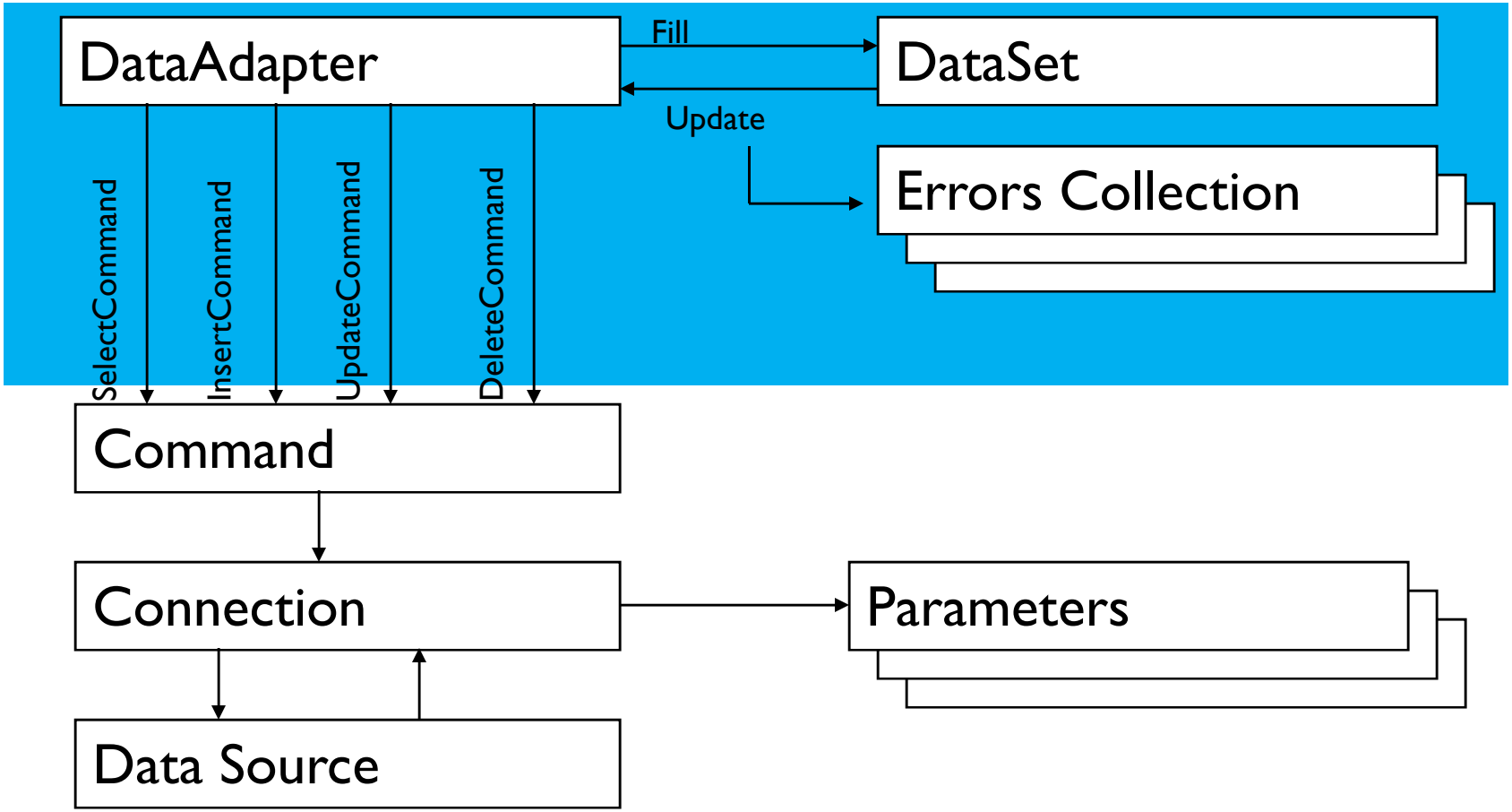
.NET Data Providers



Funksionaliteti i Data Provider



Modeli i objektit ADO.Net



Namespaces per qaje ne te dhena

- System.Data
- System.Data.Common
- System.Data.SqlClient
  System.Data.OleDB
- System.Data.SqlTypes
- System.XML
- System.XML.Schema

Shfrytezimi i Namespaces-ve

- VB.Net

```
Imports System.Data
Imports System.Data.SqlClient
Dim sqlAdp as SqlDataAdapter
```

- C#

```
using System.Data;
using System.Data.SqlClient;
SqlDataAdapter sqlAdp= new
SqlDataAdapter();
```

SQL Namespace Objects

- `using System.Data.SqlClient;`
- SqlConnection
- SqlCommand
- SqlDataReader
- SqlDataAdapter
- SqlParameter
- SqlParameterCollection
- SqlError
- SqlErrorCollection
- SqlException
- SqlTransaction
- SqlDbType

Lidhaj (Connecting) me SQL

```
using System.Data.SqlClient;

string sConnectionString =
  "Initial Catalog=dbUniversitetiAAB;
   Data Source=localhost;
   Integrated Security=SSPI;";

SqlDataAdapter sqlAdp= new
SqlDataAdapter(sConnectionString);

sqlAdp.Close();
sqlAdp.Dispose();
```

Lidhaj (Connecting) me SQL

- Lidhje me ADO.Net:

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString =
  "Integrated Security=SSPI;Initial Catalog=northwind";
conn.Open();  // Pool A is created.
```

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString =
  "Integrated Security=SSPI;Initial Catalog=pubs";
conn.Open();
// Pool B is created because the connection strings
differ.
```

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString =
  "Integrated Security=SSPI;Initial
Catalog=dbUniversitetiAAB";
conn.Open(); // The connection string matches pool A.
```

Leximi i te dhenave

- SqlCommand
  - ExecuteReader
  - ExecuteNonQuery
  - ExecuteScalar
  - ExecuteXMLReader
- SqlDataAdapter
  - DataSet

Perdorimi i "command object"

- SqlCommand
  Multiple constructors
- `New()`
- `New(cmdText)`
- `New(cmdText, connection)`
- `New(cmdText, connection, transaction)`

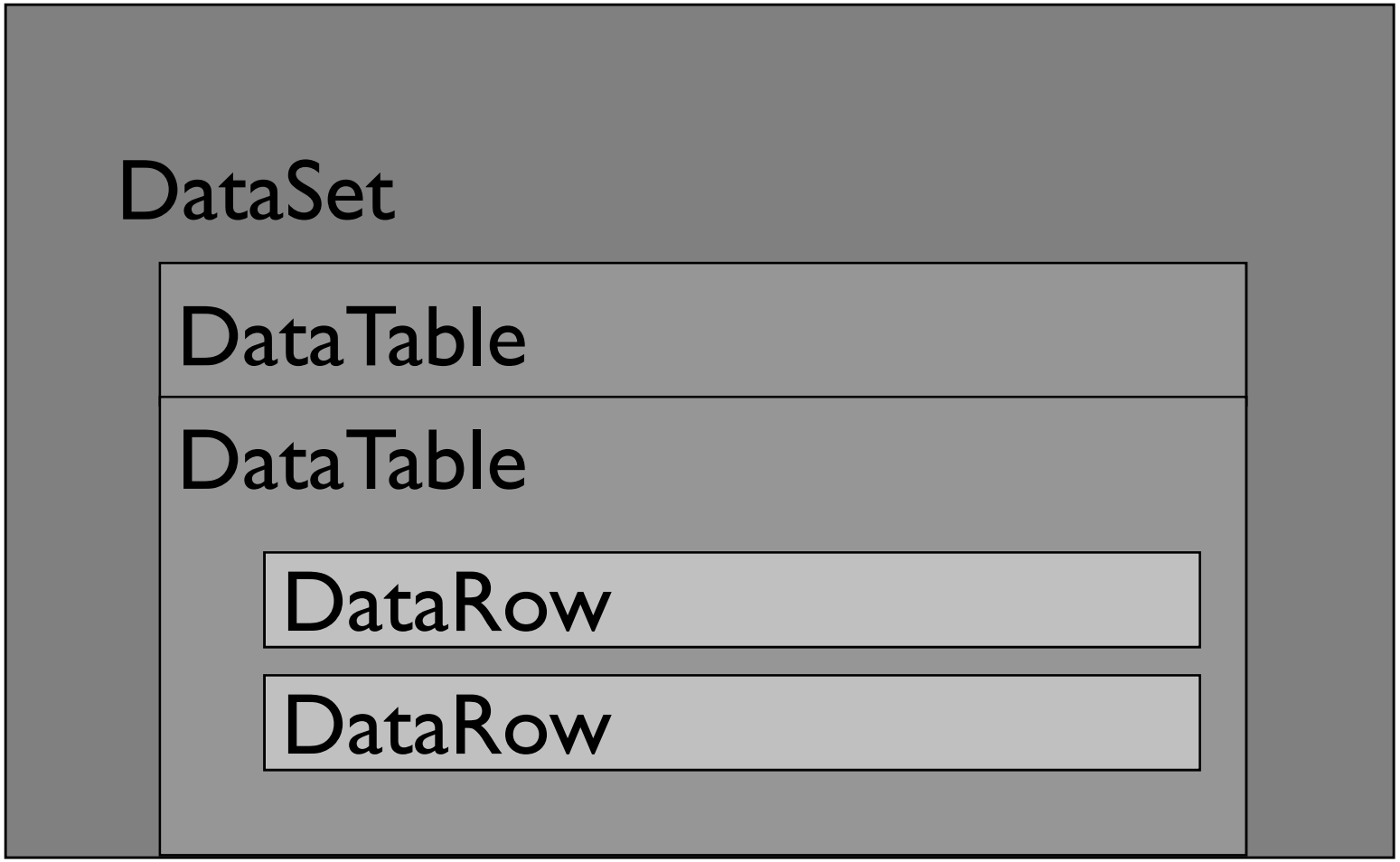
Perdorimi i "command object"

```
string sSelectQuery =
   "SELECT * FROM tblStudenti ORDER BY IDstudentit";
string sConnectionString =
   "Initial Catalog=dbUniversitetiAAB;
    Data Source=localhost;
    Integrated Security=SSPI;";
SqlConnection objConnect = new
SqlConnection(sConnectString);
SqlCommand objCommand = new     SqlCommand(sSelectQuery,
objConnect);

/*
objCommand.CommandTimeout = 15;
objCommand.CommandType = CommandType.Text;
*/

objConnect.Open();

SqlDataReader drResults;
drResults = objCommand.ExecuteReader()

drResults.Close();
objConnect.Dispose();
```

Komandat per Metodat per qasje ne te dhena

- .ExecuteReader() – Kthen DataReader
- .ExecuteNonQuery() – **Kthen # of Rreshat e Afektuara**
- .ExecuteXMLReader() – Kthen XMLReader Object per leximin e XML dokumentit
- .ExecuteScaler() – Kthen nje vlere p.sh. SQL SUM funksionin.

Sets, Tabelat dhe Rreshatat



Perdorimi i "DataTables"

Me DataTable ne mundemi te bejme:

- Insert, modify and update
- Search
- Apply views
- Compare
- Clear
- Clone and
- Copy

Qasja Indirekte në të dhëna

Të dhënave të enkapsuluara në klasë nuk mund t'u qasemi direkt nga klasat e tjera.
Të dhënave u qasemi në mënyrë **indirekte** përmes operacioneve.

- Pse qasja indirekte në të dhëna?
  - Mbron dhe mbulon të dhënat:
    - E mundshme të kontrollohen në kushte të caktuara gjatë secilës qasje në të dhëna.
    - E mundshme të ndërmerren veprime të caktuara gjatë secilës qasje në të dhëna.
  - E bën të lehtë, në të ardhmen, ndryshimin e shfaqjes së të dhënave
    - Përmes "kompenzimit" të programuar në operatorin e qasjes.
  - E bën të mundshme evitimin e alokimit të hapësirës për disa të dhëna
    - Llogaritje në vend të ruajtjes

Tipet referuese

- Objekteve u qasemi përmes referencave.
- Kur krijojmë një objekt duke instancuar, marrim një referencë tek objekti i ri.

Klasa është tip me refercë.

Objektet e instancuara nga klasat u qasemi me referenca.

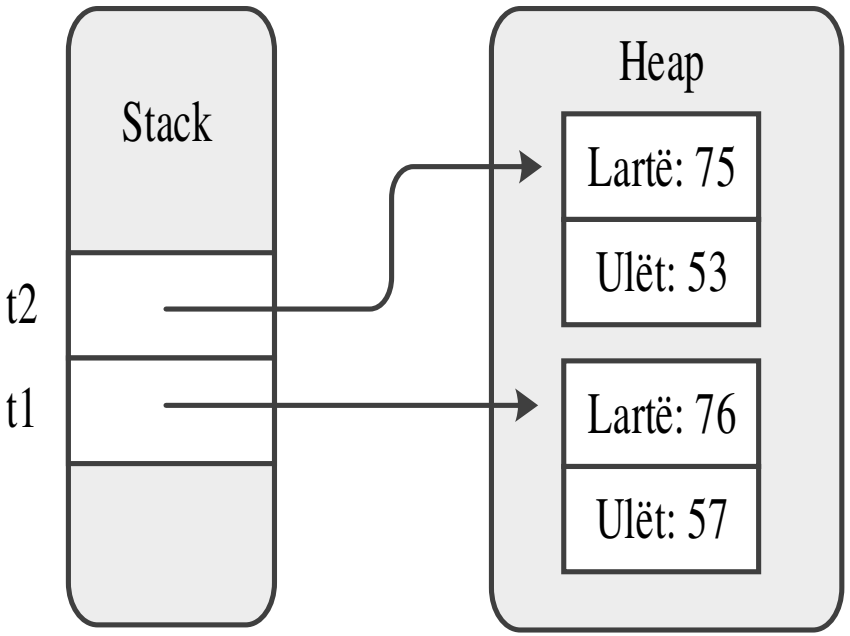
Objektet alokohen në *heap*.

Instancat e klasave trajtohen me të ashtuquajturat *semantikat referencore*.

Tipet referuse në C# korrespodojnë në pointerë në C.

Tipet referuse

- Semantikat referencore:
  - Vendosja e vlerave, pasimi i parametrave dhe **return** manipulojnë *referenca tek objektet.*
  - Heap
    - Hapsirë memorike në të cilën alokohen instancat e klasave.
    - Alokimi zë vend në momentin kur instancohet klasa.
    - Dealokimi zë vend kur objekti nuk ka ndonjë efekt në program
      - Në praktikë, kur nuk ka referenca aktive në objekt.
      - Nga një fije e ndarë e kontrollit e quajtur *Garbage Collector*.



Ilustrimi i variablave të Tipit me referencë

Shembulli i treguar në figurën e mëposhtme tregon variablat p1 dhe p2 para se të ekzekutohet linja **p1=p2** në kodin e mëposhtëm. Variablat nuk i përmbajnë vlerat por vetëm referencat tek pikat.

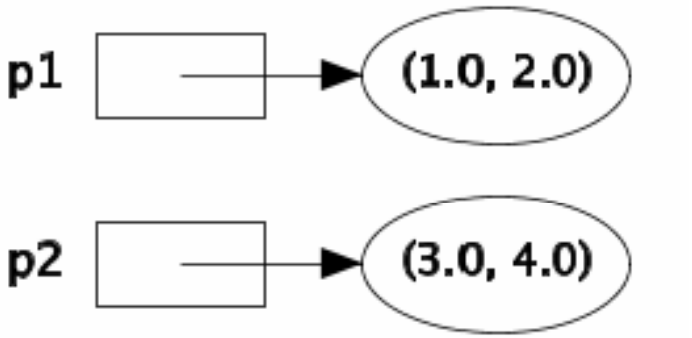


Fig. Variablat p1 dhe p2 i referohen dy pikave

```
Pika p1 = new Pika(1.0, 2.0),
p2 = new Pika(3.0, 4.0);
p1 = p2;
```

Ilustrimi i variablave të Tipit me referencë

Duke e vazhduar vendosjen e vlerës **p1=p2**, të dy pikat **p1** dhe **p2** do t'i referohen objektit të njëjtë **Pika.**

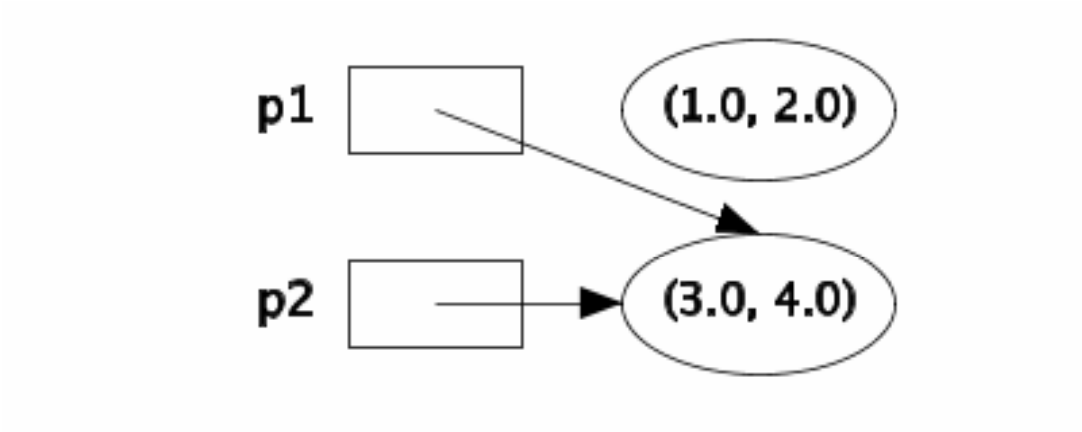


Fig. Variablat e tipit referencorë. Situata pasi të vendoset **p1=p2**

```
Pika p1 = new Pika(1.0, 2.0),
     p2 = new Pika(3.0, 4.0);
     p1 = p2;
```

Tipet me referencë në C#

- Klasat janë tipe me referencë në C#,
- Është e natyrshme të pyetet se cilat tipe në C# veprojnë si tipe me referencë, e cilat jo.
- Tipet me referencë në C#, janë si vijon:
  - Klasat
    - Stringjet
    - Vargjet (arrays)
  - Ndërfaqet
    - Të ngjashme me klasat.
    - Nuk përmbajnë të dhëna.
    - Kanë vetëm nënshkrimet e metodave.
  - Delegatët
    - Objektet delegate mund të kenë një ose më shumë metoda.
    - Përdoren kur i trajtojmë metodat si të dhëna.

Krahasimi dhe kopjimi i objekteve përmes tipeve me referencë

- Shumë diskutime dhe pyetje lidhur me krahasimin dhe kopjimin e objekteve që kanë qasje me referenca.
- Ne do të listojmë disa nga to dhe do të provojmë t'u përgjigjemi në vijim:
  1. A i krahasojmë referencat apo objektet e referuara?
  2. A e kopjojmë referencën apo objektin e referuar?
  3. Sa thellë i kopjojëm objektet që i referojnë objektet e tjera

p1 == p2 (krahason lokacionin / adresat e referencave)
p1 = p2 (e kopjon vetëm adresen, e jo objektin)

Krahasimi dhe kopjimi i objekteve përmes referencave

- Krahasimi:
  - Krahasimi referencor
    - Krahason nëse dy referenca i referohen objektit të krijuar nga ekzekutimi i njejtë i **new**
  - Krahasimi shallow (i cekët) dhe i thellë (deep)
    - Krahasim i fushave – në nivele të ndryshme
- Kopjimi
  - Kopjimi referencor
  - Kopjimi i cekët ose i thellë
    - Gjithashtu i njohur si *klonim*
    - I mbështetur nga metoda **MemberviseClone** në **System.Object**

Barazimi në C#

Të gjitha metodat e diskutuar në këtë seksion i përkasin klasës **Object**.

- **o1.Equals(o2)** - Kontrollohet barazia (a janë objektet e barabarta).
  - Me nënkuptim, *true* nëse o1 dhe o2 janë krijuar nga ekzekutimi i njëjtë **new**.
  - Mund të ri-definohet në klasën përkatëse
- **Object.RefereneceEquals(o1, o2)** - identiteti.
  - **True** nëse o1 dhe o2 që të dyja janë **null,** ose nëse janë krijuar nga ekzekutimi i njejtë i **new**.
  - **Static** – nuk mund të ridefinohet
- **Object.Equal(o1, o2)**
  - **True** nëse Object.ReferenceEquals(o1,o2), ose nëse o1.Equals(o2).
- **o1= = o2**
  - **True** nëse o1 dhe o2 janë **null**, ose nëse janë krijuar nga ekzekutimi i njejtë i **new.**
  - Operator i mbingarkuar.

Tipet me Vlerë

- Vlerat në tipet me vlerë nuk u qasemi përmes referencave.
- Në C# nuk është e mundur të u qasemi këtyre tipeve përmes referencave.
- Variablat e tipeve me vlerë përmbajnë vlerat e tyre (e jo referencat për vlerat e tyre).
- Alokohen në *Stack*.
- *Tipet me vlerë* në C# janë si vijon:
  - Tipet numerike, char, boolean, tipet e numërimeve
  - Strukturat (*char* dhe *boolean* janë të definuara si struktura në C#)

Tipet me Vlerë

Variabla e tipit me vlerë përmban vlerën e saj.

Vlerat alokohen në *stack (rafte)* të metodës ose në objektet në *heap*.

Variablat e tipeve me vlerë trajtohen me të ashtuquajturën semantikë e vlerave.

Përdorimi i tipeve me vlerë e lehtëson punën me të dhëna të përkohshme.

Semantika e vlerave:

Vendosja, kalimi i parametrave, thirrja-me-vlerë dhe return kopjon vlerat e tëra.

Stack i metodës është:

- Hapësira memorike ku alokohen të dhënat me jetëgjatësi të shkurtër.
- Parametrat dhe variablat lokale.
- Alokimi zë vend kur operacioni i thirrjes së metodës së tillë të ndodh.
- Dealokimi zë vend në kthimin e operacionit.

Ilustrimi i variablave të tipit me vlerë

- Marrim se *Pika* është e tipit me vlerë.
- Në C# kjo do të realizohet si Strukturë (*struct*).
- Në figurën e mëposhtme i shfaqim dy variabla, **p1** dhe **p2**, që përmbajnë vlera **Pika**.
- Situata në figurën e mëposhtme mund të arrihet me shoqërimin e vlerave gjatë inicimit të instancave.

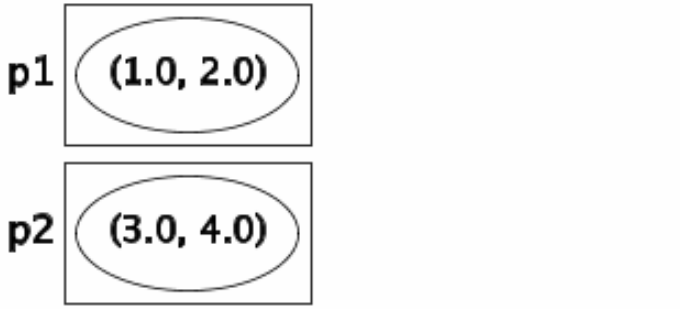


Fig. Variablat e tipit me vlerë. Situata para se të vendoset p1= p2.

```
Pika  p1 = new Pika (1.0, 2.0),
       p2 = new Pika(3.0, 4.0);
       p1 = p2;
```

Variablat p1 dhe p2 i referohen dy pikave .

Ilustrimi i variablave të tipit me vlerë

- Situata pas vendosjes së vlersë p1=p2 është si në figurën e mëposhtme:

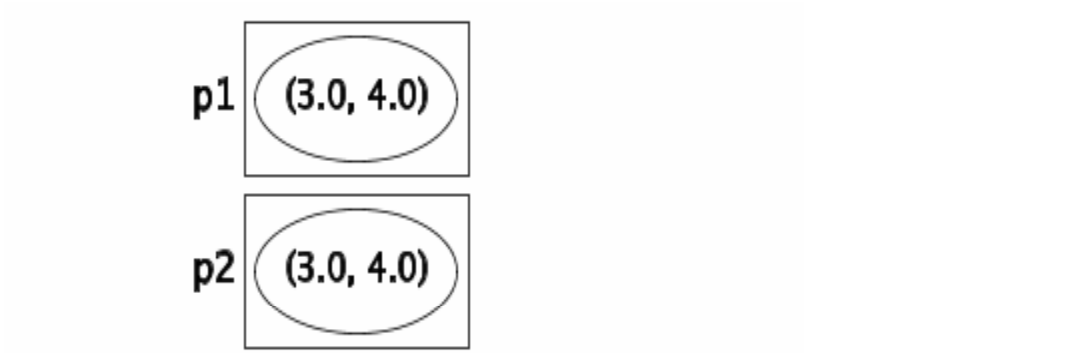


Fig. Variablat e tipit me vlerë. Situata pas vendosjes së p1=p2.

Vendosja e p1=p2 e bën kopjimin e vlerës që gjendet në p2 në objektin p1.

Operatori i krahasimit p1==p2 i krahason vlerat në objekte (krahasimi bitwise).

Strukturat në C#

- Janë të ngjashme me klasat.
- Dallimi: janë tipe me vlerë, klasat janë referenciale.
- Si në klassa, është e rekomandueshme që të programoni një ose më shumë konstruktorë në strukturë.
- Nuk mund të jetë konstruktor pa parametër .

```
using System;
public struct Pika {
private double x, y;
public Pika(double x, double y){
this.x = x; this.y = y;
}
public double MerrX(){
return x;
}
public double MerrY(){
return y;
}
public void Levize(double dx, double dy){
x += dx; y += dy;
}
public override string ToString(){
return "Pika: " + "(" + x + "," + y + ")"
+ ".";
}
}
```

Tipi Pika i realizuar përmes Strukturës.

Strukturat në C#

- Strukturat përdoren për agregim dhe enkapsulim të *disa vlerave*, të cilat dëshirojmë t'i trajtojmë si vlera, dhe për të cilat dëshirojmë të aplikojmë semantikat e vlerave.

- Në namespace-in **System**, tipet **DateTime** dhe **TimeSpan** janë të programuara si struktura.

Strukturat dhe Inicializimi

Janë disa rregulla të inicializimit të vlerave të strukturës, së paku krahasuar me klasat.

- Nuk mund të ketë inicim të fushave të instancës në strukturë

```
public struct Struktura1{
    int a=5; double d = 7.7;
}
```

Gabim! Nuk lejohet inicimi I vlerave të fushave të instancës

```
public struct Struktura1{
    int a; double d;
}
```

Vlerat e fushave të strukturës inicohet në vlera të nënkuptuar a=0, d=0.0 (automatikisht).

Strukturat dhe Inicializimi

Nuk mund të ketë konstruktor eksplicit pa parametra.

```
public struct Struktura1{
        int a; double d;

      public Struktura1()
      {
         a=1;  d=6.1;
      }
}
```

Gabim! Konstruktori pa parametra nuk lejohet në Strukturë

```
public struct Struktura1{
        int a; double d;

      public Struktura1(int a, double d)
      {
         this.a=a;  this.d=d;
      }
}
```

Ne mund të programojmë konstruktorë me parametra. Konstruktori pa parametra ekziston në mënyrë implicite.

Krahasimi i Strukturave me Klasa

Klasat	Strukturat
Tipe referencore	Tipe me vlerë
Përdoren me instancim dinamik	Përdoren me instancim statik
Pasardhës të klasës **Object**	Pasardhës të klasës **Object**
Mund të zgjerohen me ndërfaqe	Mund të zgjerohen me ndërfaqe
Mund të implementojnë një ose më shumë ndërfaqe	Mund të implementojnë një ose më shumë ndërfaqe
Mund të inicojnë fushat me inicues	Nuk mund të inicojnë fushat me inicues
Mund të kenë konstruktor pa parametra	Nuk mund të kenë konstruktor pa parametra

Vetitë – Përdorimi bazë

Vetitë lejojnë vendosjen e vlerave dhe leximin e vlerave në mënyër indirekte për variablat e instancës.

```
public class C {
 private int v;

 public C(int t){
   v = t;
 }
 public int V {
   get  {return v;}
   set  {v = value;}
 }
}
```

Klasa C me një veti të thjeshtë – për shkrim dhe lexim (getter , setter)

```
using System;
public class Client {

 public static void Main(){

   C x = new C(5);

   x.V = 7;              // aktivizon pjesën për shkrim: v merr vleren 7.
   Console.WriteLine(x.V);   // aktivizon pjesën për lexim: shkruan 7.
 }

}
```

Klienti i klasës C

Vetitë – Përdorimi i ndërlikuar

```
public class C {
  private int v;

  public C(int t) {
    v = t;
  }
  public int V {
    get  {return v*2;}
    set  {v = value+3;}
  }
}
```

Klasa C me një veti të ndërlikuar– për shkrim dhe lexim (getter , setter)

```
using System;
public class Client {

  public static voID Main(){

    C x = new C(5);

    x.V = 7;              // aktivizon pjesën për shkrim: v merr vlerën .
    Console.WriteLine(x.V);   // aktivizon pjesen per lexim: shkruan .
  }

}
```

Klienti i klasës C

Vetitë në C# - Shembull

```
class Person {
        private int ID;
        private string emri;

        //Vetia ID
        public int ID
        {
              get { return ID; } //pjesa për lexim

            //pjesa per shkrim
           // set { ID = value; }
        }
        //vetia per Emrin
        public string Emri
        {
            get { return this.emri; }
            set { this.emri = value; }
        }
}
```

Vetitë Automatike

Nga C# 3.0 është e mundur që të automatizohet definimi i atributeve për lexim dhe shkrim të vetive.

```
using System;
public class LlogariaBankare {
  // gjenerim automatik i variablave private
    public LlogariaBankare(string pronari, decimal bilanci) {
        this.Pronari = pronari;
        this.Bilanci = bilanci;
    }
    public string Pronari { get; set; }
    public decimal Bilanci { get; set; }
    public override string ToString()   {
        return Pronari + " ka gjithsej " + Bilanci + " euro";
    }
}
```

Vetitë automatike i kontribuojnë C# 3.0 *shtresës së komoditetit* në krye të termeve ekzistuëse.

Inicimi i Objekteve përmes Vetive

Nga C# 3.0 vetitë mund të përdoren së bashku me konstruktorët për të lehtësuar krijimin e objekteve të reja.

```
public class LlogariaBankare {
    // gjenerimi automatik i variablave private te instances
    // konstruktori i nenkuptuar pa parametra
    public string Pronari { get; set; }
    public decimal Bilanci { get; set; }
    public override string ToString() {
        return Pronari + " ka total " + Bilanci + " euro.";
    }
}
public class Klienti{
    public static voID Main() {
        LlogariaBankare ba1 = new LlogariaBankare {Pronari = "Gerti", Bilanci = 250},
                        ba2 = new LlogariaBankare {Pronari = "Gena", Bilanci = 1200};
        Console.WriteLine(ba1);
        Console.WriteLine(ba2);
    }
}
```

Përmbledhje e vetive në C#

```
Modifikatori  tipi-kthyes  emri-vetise  {
            get {trupi-i-get"}
            set {trupi-i--set}
 }
```

- Karakteristikat e vetive
  - Ofrojnë qasje: për lexim, shkrim dhe shkrim-lexim
  - Ty dyja vetitë e instancës dhe të klasës (statike) kanë kuptim
  - Konfigurimet e vetive paraqiten në të majtë të vendosjes së vlerës, dhe në ++ ose –
  - Vetitë triviale mund të definohen automatikisht
  - Vetitë duhet të jenë të shpejta, dhe pa ndonjë efekt anësor

Vetitë në C# shpesh kanë emrin e njejtë me anëtarët privat.
Emri i vetisë shënohet me shkronjën e parë të madhe, ndërsa emri i anëtarit jo.

Indeksat në C#

Indeksat mundësojnë qasjen në të dhëna përmes notacionit të përdorur në vargje.

```
using System;
public class A {
    private double d, e, f;
    public A(double v){
        d = e = f = v;}
    public double this[int i]
    {   get
        {   switch (i)
            {   case 1: { return d; }
                case 2: { return e; }
                case 3: { return f; }
                default: throw new Exception("Gabim!");
            }
        } set
        { switch (i)
            { case 1: { d = value; break; }
                case 2: { e = value; break; }
                case 3: { f = value; break; }
                default: throw new Exception("Gabim!");}}}
    public override string ToString()
    {
        return "A: " + d + ", " + e + ", " + f;
    }
}
```

Një klasë me Indeksa.

Indeksat në C#

Klienti i klasës A përdor indeksa të klasës A.

```
using System;
class B
{
    public static voID Main()
    {
        A a = new A(5);
        double d;

        a[1] = 6; a[2] = 7.0; a[3] = 8.0;

        d = a[1] + a[2];
        Console.WriteLine("a: {0}, d: {1}", a, d);
    }
}
```

Klasa B – shfrytëzon indeksat e klasës A

Vargjet Asociative

Vargu asociativ është një varg i cili lejon indeksimin në kuptimin e objekteve arbitër, jo vetëm integer(numra të plotë)

- Vargjet asociative lidhin bashkësinë e objekteve (objektet indeksuese, qelësat) në një bashkësi tjetër objektesh (objekti element)

```
public class A {
    private double d, e, f;
    public A(double v){
        d = e = f = v;}
    public double this[string str]
    {   get
        {   switch (str)
            {   case “d”: { return d; }
                case “e”: { return e; }
                case “f”: { return f; }
                default: throw new Exception("Gabim!");}}
         set
        { switch (str)
            {   case “d”: { d = value; break; }
                case “e”: { e = value; break; }
                case “f”: { f = value; break; }
                default: throw new Exception("Gabim!");}}}
    public overrIDe string ToString()
    {
        return "A: " + d + ", " + e + ", " + f;
    }
}
```

Klasa A e indeksuar me **string.**

Vargjet asociative në C# implementohen përmes hashtable në dictionaries (fjalorë).

Përmbledhje e indeksave në C#

modifikatori tipi-kthyes ***this***[lista formale e parametrave]
get {trupi i get}
set { trupi i set}

- Karakteristikat e indeksave:
  - Ofrojnë qasje me indeksim vetëm për lexim, vetëm-shkrim, ose shkrim-lexim tek të dhënat e objekteve
  - Indeksat mund të jenë vetëm anëtarë të instancave, jo statik
  - Indeksat mund të bazohen në një, dy ose më shumë parametra formal
  - Indeksat mund të jenë të mbingarkuar
  - Indeksat nuk duhet të mos kenë efekte të panevojshme anësore
  - Indeksat duhet të jenë të shpejtë

Metodat në C#

Metodat e instancës së klasës C kanë për tendencë kryerjen e operacioneve në instancën e C.
Operacione të tilla e modifikojnë gjendjen e instancës së C, dhe gjithashtu mund të ekstraktojnë të dhëna nga instancat e C.

Forma e përgjithshme e metodës:

Modifikatori tipi-kthyes emri-metoteds (lista-e-parametarve-formal) { statements }

Shembull:

```
public string Studenti(string emri, string mbiemri)
        {
            // ...implementimi i metodes me dy parametra
            string rezultati = emri + mbiemri;
            return rezultati;
        }
```

Variablat lokale në metoda

Variablat e deklaruara brenda bllokut të metodës quhen *varibla lokale*.

- Variablat lokale
  - Mund të deklarohen kudo në bllok
    - Nuk janë të domosdoshme në pjesen inicuëse të bllokut
  - Mund të inicohen nga inicues
  - Mund të deklarohen në mënyrë alternative pa inicues
    - Nuk caktohet vlerë e paracaktuar për variablat lokale
      - Ndryshe nga variablat anëtare të klasës dhe instancës të cilave u jepen vlera të nënkuptuara
    - Kompajleri do të ankohen nëse variabla lokale referohet pa i'u vendosur vlerë

Parametrat

Parametrat aktual shfaqen në thirrje.
Parametrat formal paraqiten në deklarimin e metodës.
Në përgjithësi, prametrat aktual janë shprehje të cilat evaluohen në argumentet.
Varësisht nga lloji i parametrave, argumentet shoqërohen disi me parametrat formal.

- Ka katër lloje të ndryshme të parametrave në C#
  - Parametrat me vlerë
    - Modi i nënkuptuar i pasimit të parametrave, pa përdorim të modifikatorëve
  - Parametra me referencë
    - Të specifikuara me **ref**
  - Parametra dalës
    - Të specifikuar me **out**
  - Vargje parametrash
    - Të specifikuar me modifikatorin **params**

Parametrat me vlerë

Parametrat me vlerë përdoren si vlera hyrëse në metoda.

- Pasimi i parametrave me vlerë
  - Parametri formal korespondon me variablën lokale
  - Parametri formal inicohet me argumentin koresponduës (vlerën aktuale të parametrit)
    - Krijohet një *Kopje* e argumentit për parametrin formal
  - Konvertimi implicit mund të zë vend
  - Vendosja e vlerave në parametër është e mundshme por pa ndonjë efekt jashta metodës

Vendosja e vlerave në parametrat formal nuk ndikon në vlerën aktuale koresponduëse të parametrit.

Parametrat me referencë

Parametrat me referencë mund të përdoren si vlera hyrëse dhe dalëse në metodat.

- Pasimi i parametrave me referencë
  - Argumenti korespondues duhet të jetë variabël, dhe duhet të ketë vlerë
  - Tipet e parametrave formal dhe të argumentit duhet të jenë identik
  - Parametri formal bëhet me emër tjetër (alias) të argumentit
  - Parametri aktual duhet të prefiksohet me fjalinë kyçe **ref**

```
public static void RefNdryshoVendet(ref int a, ref int b)
        {
            a = a + b;
            b = a - b;
            a = a - b;
        }
```

Parametrat dalës

Parametrat dalës përdoren për outpute(dalje) në metoda. Metoda vendos vlera në parametrat dalës.

- Pasimi i parametrave dalës
  - Argumenti korespondues duhet të jetë variabël
  - Argumenti korespondues nuk ka nevojë të ketë ndonjë vlerë fillimisht
  - Parametri formal do të duhej/duhet që të merr vlerë nga metoda
  - Parametri formal bëhet një emër tjetër (alias) i argumentit
  - Tipet e parametrit formal dhe argumentit duhet të jenë identike
  - Parametri aktual duhet të prefiksohet me fjalën kyçe **out**

```
public void Mbledhja(int v1, int v2, int v3, out int v)  {
    v = v1 + v2 + v3;
}
```

v – është parametër dalës.
Përdoret si alternativë për të kthyer vlerën v1+v2+v3.

Përdorimi i ref dhe out parametrave

Sa janë të dobishëm parametrat me referencë dhe vlerë në programimin e orientuar në objekte?

- Metoda publike me dy ose më shumë pjesë dalëse
  - Përdorimi i një numri të **out** parametrave
  - Ndrysho objektet e pasuara si parametra me vlerë
  - Agrego me pjesët dalëse në objekt dhe kthej ato
  - Vendos vlerat dalëse në variabla të instancës të cilat pastaj mund të qasen nga thirrësit

parametrat **ref** dhe **out** janë relativisht të rrallë në libraritë standarde të C#

Parametrat vargje

Parametri varg është parametër formal i tipit varg i cili mund të marrë zero, një ose më shumë parametra.

- Pasimi i parametrave vargje
  - Elemetet e parametrave vargje pasohen me vlerë
  - Parametri varg duhet të jetë i fundit në listën e parametrave formal
  - Parametri formal duhet të jetë varg një dimensional
  - Argumentet që përcjellin vlerat ordinare, **ref** dhe **out** vendosen në një varg alokues të ri të objekteve
  - Argumentet në mënyrë implicite konvertohen në elemente të tipit varg (array)

Parametrat **ref** dhe **out** janë relativisht të rrallë në libraritë standarde të C#.

Extension Metodat

Nga C# 3.0 extension metoda definon metodë të instancës në një klasë ekzistuese pa ndryshuar definimin e klasës.

- Extension metoda
  - Zgjeron tipin e parametrit të parë
  - Definohet si metodë statike me modifikatorin **this** në parametrin e parë
  - Duhet të definohet në klasë statike
  - Nuk mund t'u qaset variablave private të instancës në klasën zgjeruëse
  - Thirret prapa skenës, duke u përkthyer në thirrje të metodës statike

Përmbledhje e Metodave, Vetitive dhe Indeksave

Regulat për përdorimin e metodave, vetive dhe indeksave

- Vetitë
  - Për leximin dhe ekstraktimin e variablave të instancës/klasës
  - Për shkrimin dhe vendosjen e variablave të instancës/klasës
  - Për llojet e tjera të qasjes në të dhëna që nuk konsumojnë shumë kohë gjatë llogaritjes
- Indeksat
  - Sikurse vetitë
  - Përdoren kur është natyrale që të dhënat të qasen në bazë të indeksit – notacion të vargjeve – në vend të emrave të thjeshtë
  - Përdoren si notacion sipërfaqësor për shoqërimin në varg
- Metodat
  - Për të gjitha operacionet e tjera që enkapsulojnë llogaritjen në të dhëna të klasës