.NET projekti, tipovi podataka, ključne riječi

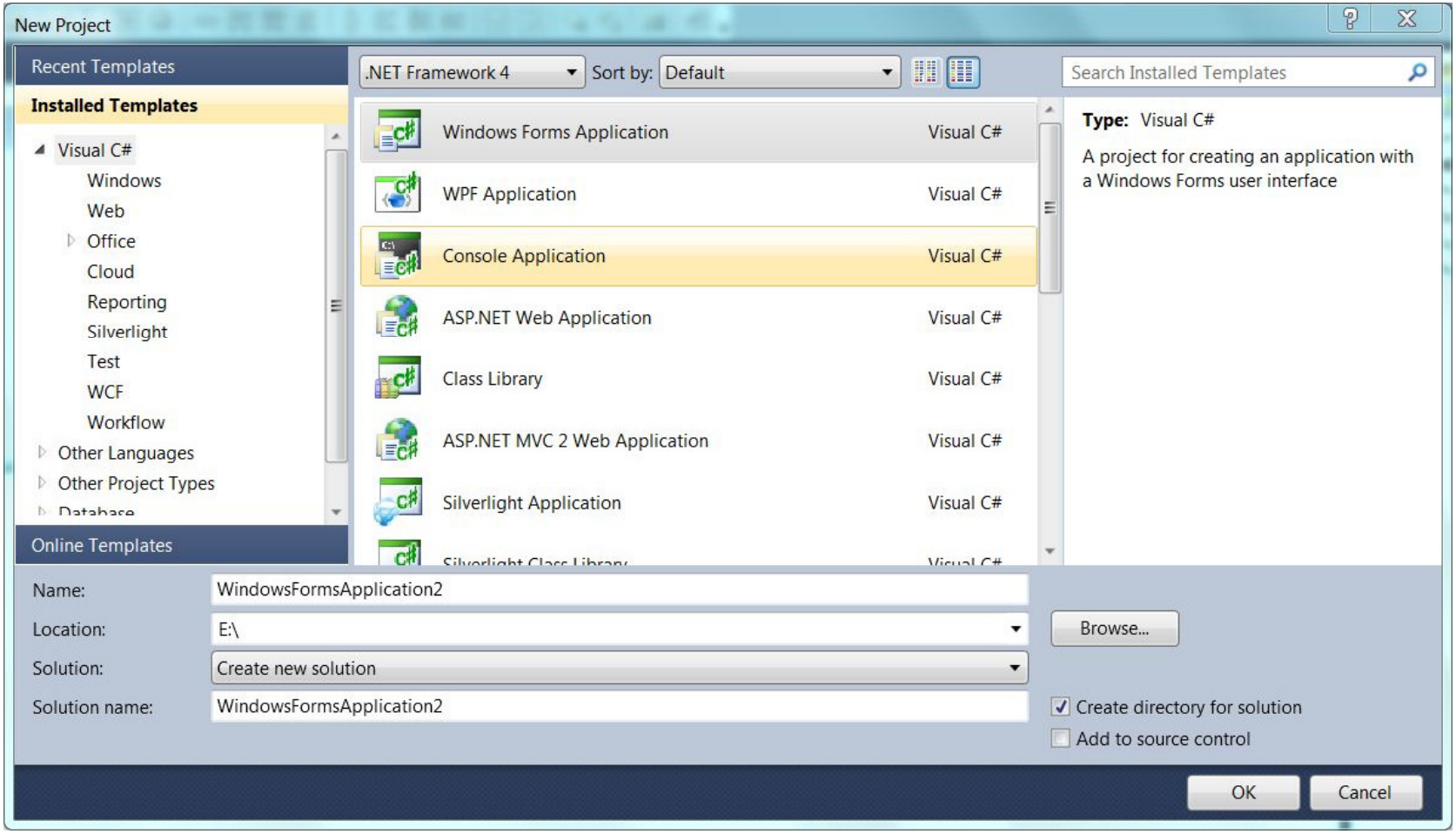
Maja Štula

Ak. God. 2011/2012

Visual Studio

- Visual Studio je Microsoftova aplikacija za razvoj aplikacija tzv. IDE (Integrated Development Environment) (ZendStudio (PHP), Eclipse (Java, itd.), ...).
- IDE aplikacije sadrže alate koji programerima omogućavaju i olakšavaju razvoja aplikacije na jednom mjestu, od osnovnih poput *editora* za pisanje koda, *kompajlera*, *debugera* i sl., do složenijih poput grafičkog sučelja za razvoj aplikacija, alata za automatsko nadopisivanja koda, itd.

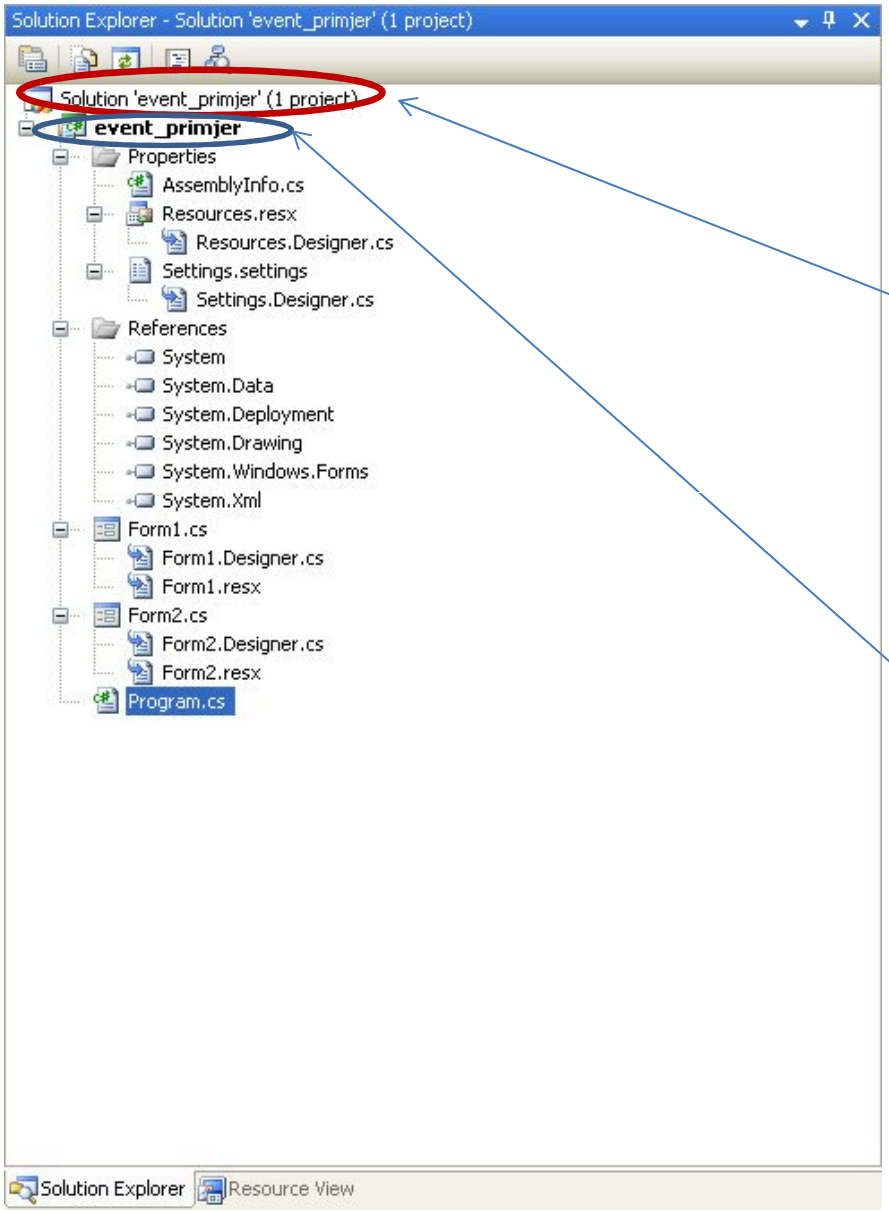
Visual Studio



Visual Studio

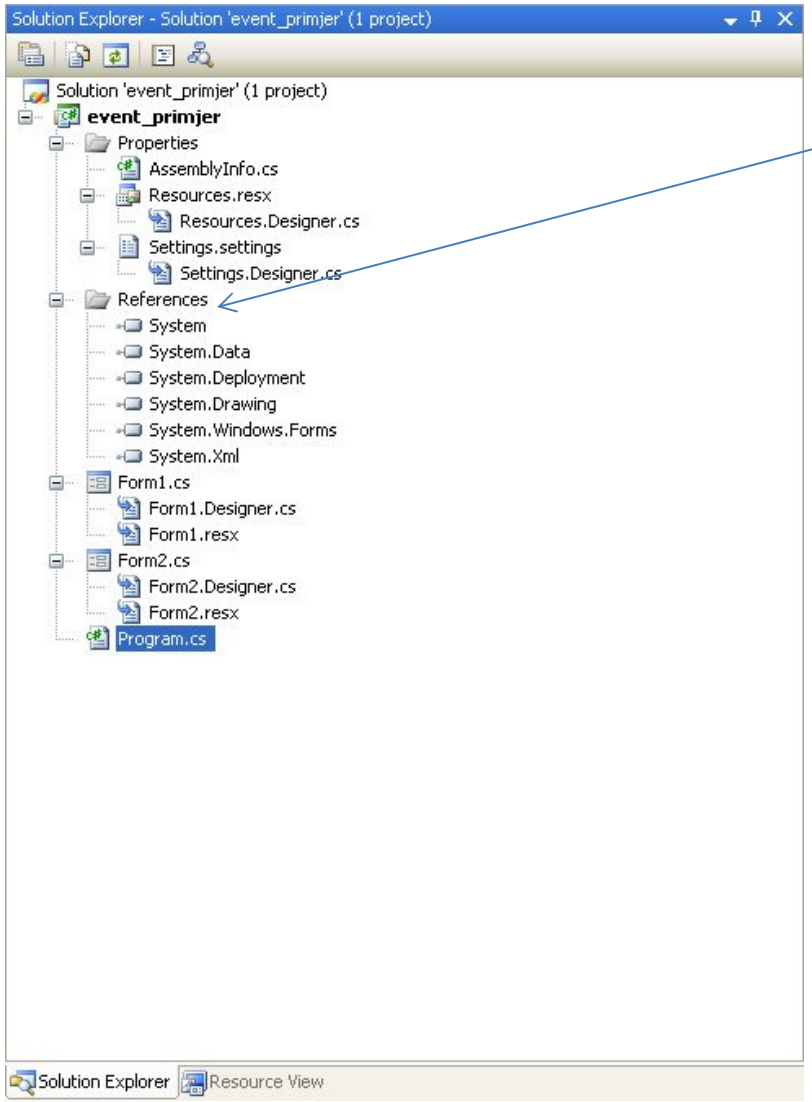
- Na prethodnom slajdu je prikazan jedan prozor u Visual Studiju koji omogućava korištenje predložaka.
- Predlošci su mehanizam koji ubrazavaju razvoj aplikacije tako što je osnovni kostur aplikacije (osnovni kod, osnovne datoteke) sadržan u predlošku, izborom određenog predloška (određenog tipa aplikacije) se generira osnovni kostur aplikacije i programer onda samo nadograđuje postojeći kostur.

Visual Studio



- Osnovna organizacijska jedinica je tzv. projektno rješenje (*solution*).
- *Solution* u Visual Studiju sadrži projekt (ili više projekata) i odnose (*dependencies*) među sadržanim projektima.
- *Solution* je kontejner projekata.
- Projekt je kontejner za datoteke koda i omogućava kompajliranje koda u izvršne datoteke.

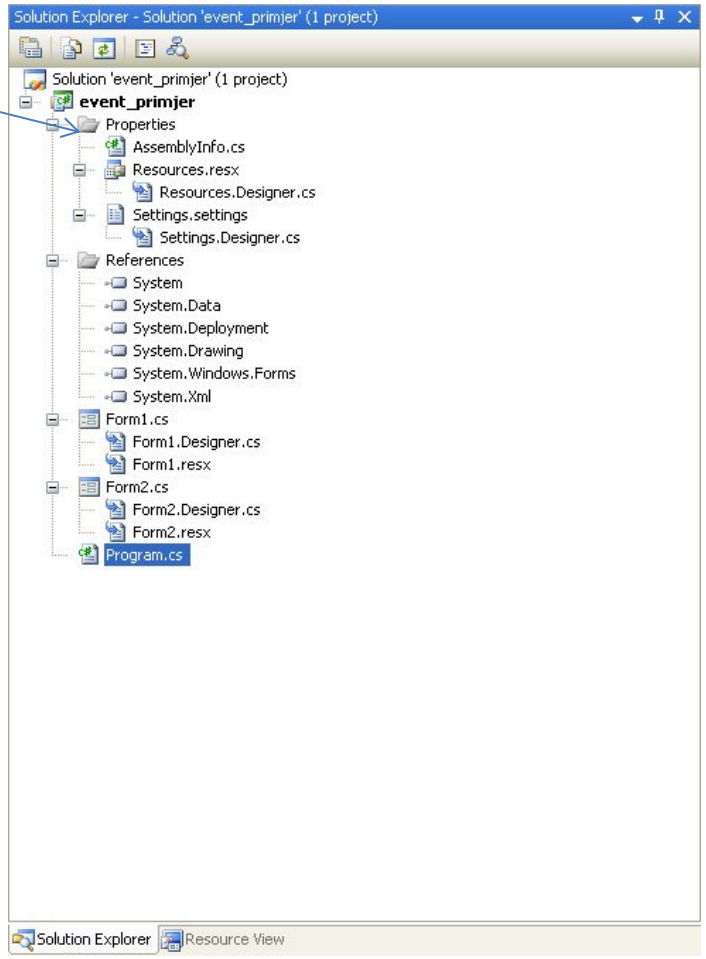
Visual Studio



- Reference projekta su poveznice prema vanjskim komponentama koje su uključene u projekt. Komponente mogu biti .NET assembliji, COM komponente i sl. Ukoliko komponenta nije uključena kao referenca u projekt ne može se koristiti.

Visual Studio

- Svojstva (*properties*) projekta sadrže svojstva projekta poput imena assemblija (u AssemblyInfo.cs [assembly: AssemblyTitle("event_primjer")]).
- Neka se svojstva odnose na cijeli projekt dok se neka odnose samo na Debug verziju ili samo na Release verziju.



OSNOVNI POJMOVI, KLJUČNE RIJEČI I SL. U C# PROGRAMSKOM JEZIKU

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

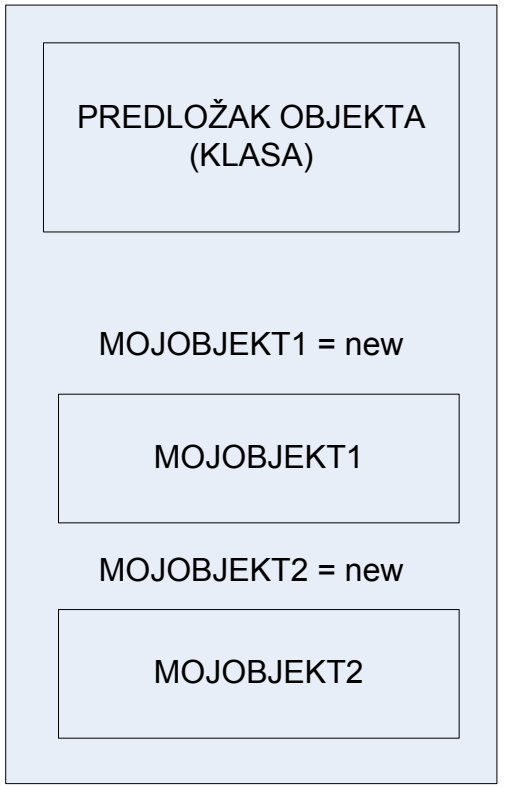
Objektno-orijentirano programiranje

Predložak objekta

```
class MojaKlasa
{
    // metode,
    // svojstva (properties),
    // polja (fields),
    // događaji, delegati,
    // ugniježđene klase
}
```

Instanca objekta

```
MojaKlasa  mojobjekt1 = new MojaKlasa();
MojaKlasa  mojobjekt2 = new MojaKlasa();
```



Klasa

- Klasa se deklarira s ključnom riječi *class*. Objekt klase se stvara ključnom riječi new.
- U C# je dozvoljeno samo jednostruko nasljeđivanje. Klasa može implementirati više sučelja.

Inheritance	Example
None	`class ClassA { }`
Single	`class DerivedClass: BaseClass { }`
None, implements two interfaces	`class ImplClass: IFace1, IFace2 { }`
Single, implements one interface	`class ImplDerivedClass: BaseClass, IFace1 { }`

Pristup

- Ključne riječi *public*, *protected*, *internal* ili *private* koriste se kao modifikatori za postavljanje mogućnosti pristupa članu.
- Modifikatori pristupa ne mogu se koristiti uz imenske prostore jer imenski prostori nemaju ograničenja pristupa.
- Modifikatori pristupa
  - *public* – pristup nije ograničen.
  - *protected* – pristup je ograničen unutar klase ili tipova izvedenih iz promatrane klase.
  - *internal* – pristup je ograničen na trenutni assembly.
  - *protected internal* – pristup je ograničen na trenutni assembly ili tipove izvedene iz promatrane klase.
  - *private* – pristup je ograničen na klasu ili tip u kojoj se nalazi član.
- Uvijek se može koristiti samo jedan tip modifikatora, osim kombinacije *protected internal* koja je dozvoljena.

Parcijalna klasa

- *partial* – ključna riječ omogućava definiranje dijela klase, strukture ili sučelja unutar imenskog prostora.
- Svi dijelovi moraju koristiti ključnu riječ *partial*.
- Svi dijelovi trebaju biti na raspolaganju prilikom kompajliranja i svi dijelovi trebaju imati iste mogućnosti pristupa (*public*, *private*, …).
- Modifikator partial se ne može koristiti s delegatima ili sa enumeriranim deklaracijama.

```
public partial class Employee
{ public void DoWork() { } }
public partial class Employee
{ public void GoToLunch() { } }
```

Enumerirani podaci

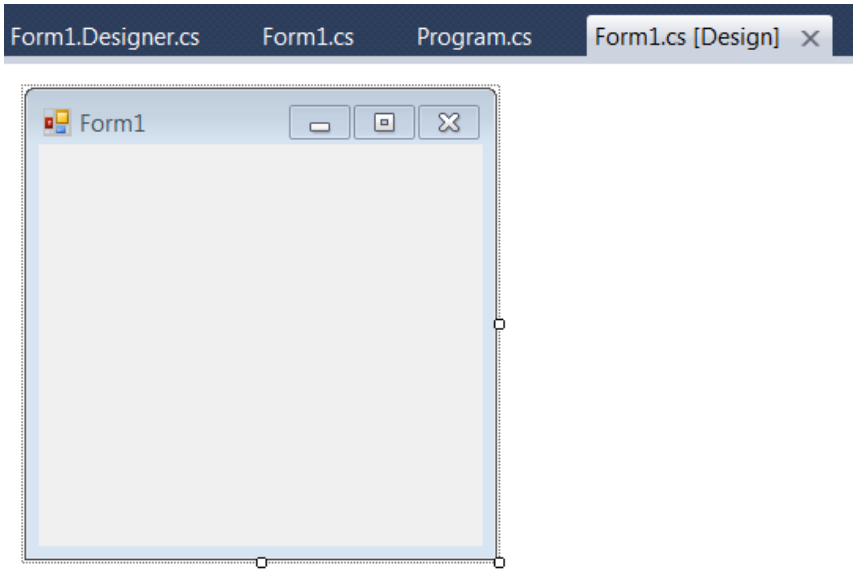
- Enumeracija je korisnički definiran tip koji se sastoji od skupa imenovanih konstanti koje se zovu enumeratori. Svaki enumerator ima vrijednost koja može biti tipa byte, short, int, long (cjelobrojni tip podatka osim char tipa) (npr. FormWindowState svojstvo klase Form).
- Defaultne vrijednosti enumeratora su *int* pri čemu prvi enumerator poprima vrijednost 0, a svaki sljedeći vrijednost uvećanu za 1.
- Enumerator se može definirati:

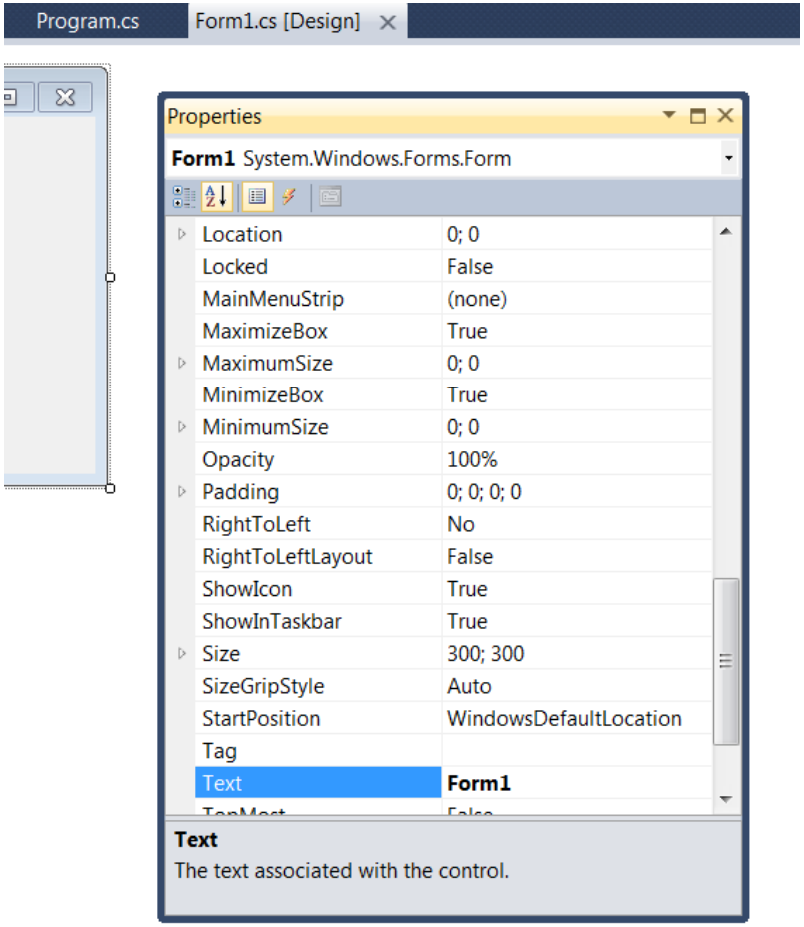
  enum day2 {sun, mon};

Design mod

- Design mod u Visual Studiu pruža mogućnost rada sa aplikacijom u tzv. design modu kada se kod aplikacije grafički vizualizira i prikazuje, a programer onda može razvijati aplikaciju “grafički”.
- Naravno treba se raditi o kodu koji je moguće vizualizirati kao što je npr. grafičko sučelje aplikacije. Ovo je još jedan mehanizam u Visual Studiju koji olakšava razvoj aplikacije.

Design mod


Form1.Designer.cs
Form1.cs
Program.cs
Form1.cs [Design]
Form1


Program.cs
Form1.cs [Design]
Properties
Form1 System.Windows.Forms.Form
Location 0; 0
Locked False
MainMenuStrip (none)
MaximizeBox True
MaximumSize 0; 0
MinimizeBox True
MinimumSize 0; 0
Opacity 100%
Padding 0; 0; 0; 0
RightToLeft No
RightToLeftLayout False
ShowIcon True
ShowInTaskbar True
Size 300; 300
SizeGripStyle Auto
StartPosition WindowsDefaultLocation
Tag
Text Form1
Text
The text associated with the control.

Prenošenje argumenata preko reference

- Ključna riječ *ref* omogućava prenošenje argumenata preko reference. Sve promjene na parametru koji je prenesen u metodu biti će vidljive i u kodu odakle je metoda pozvana. I definicija metode i poziv metode moraju koristiti ključnu riječ *ref*.

```
class RefExample
{
        static void Method(ref int i) { i = 44; }
        static void Main() {
                int val = 0; Method(ref val); // val is now 44
                        }
 }
```

- Argument koji se prenosi po referenci mora prethodno biti inicijaliziran za razliku od prenošenja reference ključnom riječi *out*.
- Svojstva (*properties*) se ne mogu prenositi preko reference.

Prenošenje argumenata preko reference

- Ključna riječ *out* omogućava prenošenje argumenata preko reference. Sve promjene na parametru koji je prenesen u metodu biti će vidljive i u kodu odakle je metoda pozvana. I definicija metode i poziv metode moraju koristiti ključnu riječ *out*
- Varijabla koja se prenosi kao *out* argument ne treba biti inicijalizirana, ali u metodi joj treba biti dodijeljena vrijednost.

```
class RefExample
{
        static void Method(out int i) { i = 44; }
        static void Main() {
                int val; Method(out val); // val is now 44
                        }
 }
```

- Svojstva (*properties*) se ne mogu prenositi preko reference.

Ključna riječ *static*

- Statička klasa sadrži samo statičke članove.
- Nije moguće kreirati instancu statičke klase.
- Statičke metode, polja, svojstva ili događaji koji se nalaze u klasi koja nije statička mogu se pozvati (preko imena klase) bez kreiranja instance klase.
- Kreirane instance klase ne mogu se koristiti za pristup statičkim članovima te klase.

Statička klasa

- Statičke klase se definiraju korištenjem ključne riječi *static*.

```
static class CompanyInfo
{
    public static string GetCompanyName()
    { return "CompanyName"; }
    public static string GetCompanyAddress()
    { return "CompanyAddress"; } //...
}
```

- Nije moguće instancirati statičku klasu korištenjem ključne riječi *new*. Statička klasa se automatski učitava od strane .NET framework CLR-a kada se učita program ili imenski prostor koji sadrži tu klasu.
- Statička klasa se koristi za organizaciju metoda koje nisu vezane uz pojedinačnu instancu objekta. Osim toga metode iz statičke klase pozivaju se bez kreiranja objekta pa su brže. Primjer .NET framework statičke klase je klasa *Math* u imenskom prostoru *System*.

Zatvorena klasa

- Zatvorena klasa se ne može nasljeđivati. Klasa se deklarira kao zatvorena korištenjem ključne riječi *sealed* u definiciji klase.

  public sealed class D {}

- Članovi klase u izvedenoj klasi koji prelaze preko (*override*) virtualnih članova osnovne klase mogu se zatvoriti. Time se poništava virtualnost za sljedeće izvedene klase.

  override ↔ overload

Virtualna metoda

- Ključna riječ *virtual* modificira metodu, svojstvo, događaj i omogućava da se u izvedenoj klasi pređe preko njih.
- Kada se pozove virtualna metoda provjerava se da li izvedena klasa svojom metodom prelazi preko (*override*) virtualne metode bazne klase. Ako ne, poziva se metoda iz bazne klase.
- Podrazumijeva se (default) da su metode ne-virtualne (non-virtual) ukoliko se eksplicitno ne navede modifikator *virtual* i takve se metode ne mogu *override-ati*.
- Modifikator *virtual* se ne može koristiti zajedno sa modifikatorima *static*, *abstract* i *override*.

Apstraktna klasa

- Modifikator *abstract* koristi se sa klasom, metodom, svojstvom, događajem. Uz klasu se koristi ukoliko je klasa namijenjena isključivo nasljeđivanju.

```
abstract class ShapesClass { abstract public int Area(); }
class Square : ShapesClass {
                int x, y;
                public override int Area() { return x * y; } }
```

- Ne može se kreirati instanca (objekt) apstraktne klase.
- Apstraktna metoda može se definirati samo u apstraktnoj klasi. Ne sadrži implementaciju i to je virtualna metoda.
- Implementacija apstraktne metode radi se u izvedenoj klasi *override-anjem*.

Svojstva (*properties*)

- Svojstva su imenovani članovi klase, strukture i sučelja.
- Svojstva su mehanizam čitanja, zapisivanja ili proračunavanja vrijednosti privatnih članova kroz pristupnike (*accessors*).
- Za razliku od članova klase svojstva se ne klasificiraju kao varijable pa ih nije moguće prenositi po referenci (*ref* ili *out* modifikator).
- Svojstva deklarirana sa modifikatorom *static* su statička svojstva i ne mogu se vezati uz instancu klase. Ukoliko nisu statična onda su to svojstva instance klase.

Svojstva (*properties*)

- Pristupnici svojstva sadrže naredbe za dohvaćanje ili postavljanje svojstva. Deklaracija pristupnika može sadržavati pristupnik *get*, ili *set* ili oba.
- Tijelo pristupnika nalikuje metodi.
- *get* pristupnik treba vratiti vrijednost tipa svojstva.

```
Class Employee
{
        private string name;
        public string Name
        {
           get { return name; }
        }
}
Employee e1 = new Employee();
Console.Write(e1.Name); // The get accessor is invoked here
```

- *get* pristupnik mora završiti ili sa *return* naredbom ili sa *throw* naredbom.

Svojstva (*properties*)

- Pristupnik *set* je sličan metodi povratnog tipa void. Koristi implicitni parametar istog tipa kao i svojstvo koji se označava s ključnom riječi *value.*

```
Class Employee
{
        private string name;
        public string Name
        {
           get { return name; }
           set { name = value; }
        }
}
e1.Name = "Joe"; // The set accessor is invoked here
```

- Implicitni parametar (*value*) ne može se koristiti za lokalne varijable u set pristupniku.

Svojstva (*properties*)

- Svojstva se klasificiraju prema pristupnicima.
- Tako svojstva koja imaju samo *get* pristupnik su *read-only* svojstva i njima se ne može dodijeliti vrijednost.
- Svojstva koja imaju samo *set* pristupnik su *write-only* svojstva.
- Svojstva koja imaju oba pristupnika su *read-write* svojstva.
- Kod deklaracije svojstva oba pristupnika trebaju biti deklarirana unutar tijela svojstva tj. {}.
- Svojstvo je član koji pruža pristup karakteristikama objekta ili klase kao što su npr. boja forme, duljina stringa i sl. Sintaksa rada sa svojstvima je slična kao i sintaksa rada s podatkovnim članovima klase, ali bez direktnog pristupa. Pristup svojstvima ide preko pristupnika.

  Primjer

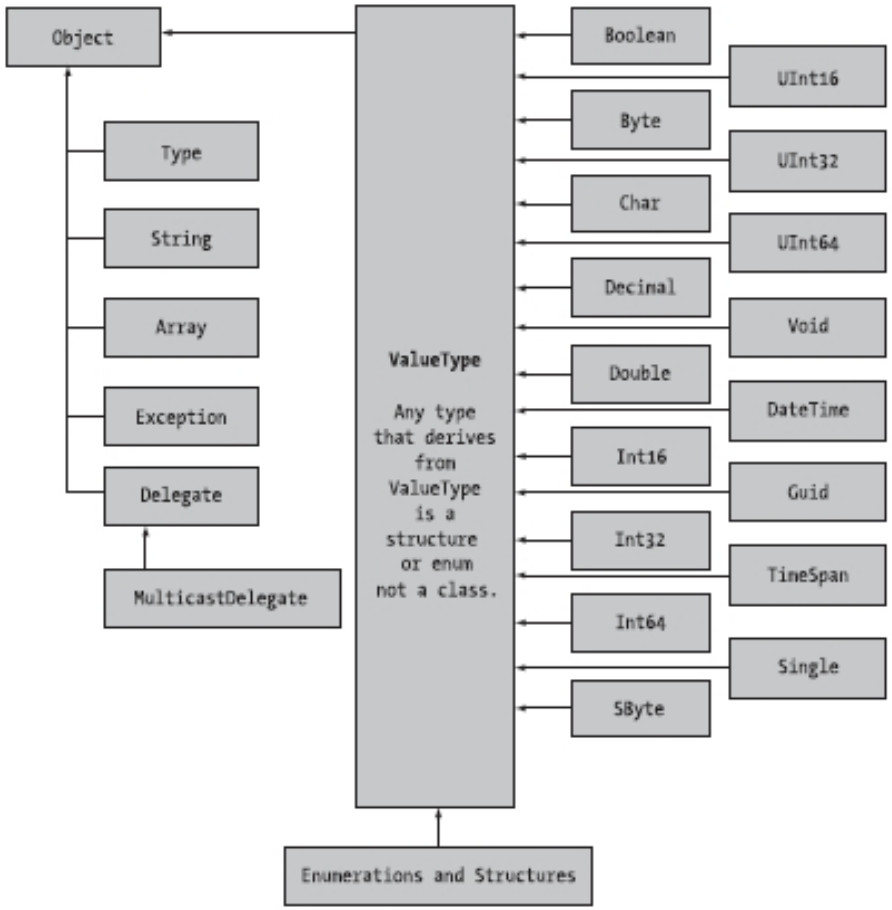
Object

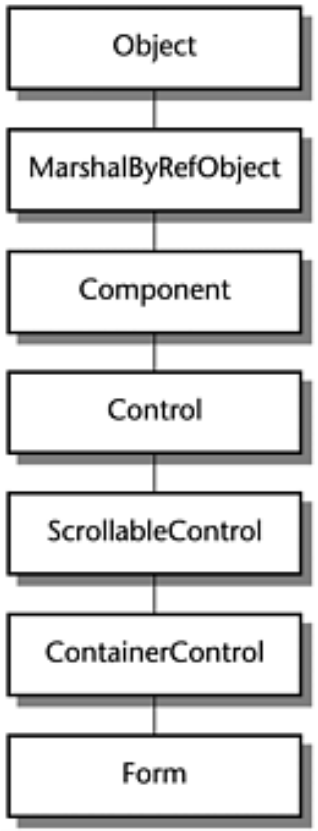
- Ključna riječ *object* je drugo ime za klasu *Object* u .NET frameworku. To je bazna klasa svih klasa u .NET frameworku. S obzirom da sve klase svih tipova, podatkovnih ili referentnih, nasljeđuju klasu *Object* varijabli tipa *object* može se pridijeliti bilo koja vrijednost.
- Kada se varijabla podatkovnog tipa konvertira u objekt kaže se da je uokvirena (*boxed*). Kada se varijabla tipa objekt konvertira u varijablu podatkovnog tipa kaže se da je *unboxed*.

Object

```
using System;
    class SampleClass
    { public int i = 10; }
    class MainClass
    {
     static void Main()
    { object a;
       a = 1;
       Console.WriteLine(a);
       Console.WriteLine(a.GetType());
       Console.WriteLine(a.ToString());
       a = new SampleClass();
       SampleClass classRef;
       classRef = (SampleClass)a;
       Console.WriteLine(classRef.i);  }
    }
```

Object


Object
Type
String
Array
Exception
Delegate
MulticastDelegate
ValueType
Any type that derives from ValueType is a structure or enum not a class.
Enumerations and Structures
Boolean
Byte
Char
Decimal
Double
Int16
Int32
Int64
SByte
UInt16
UInt32
UInt64
Void
DateTime
Guid
TimeSpan
Single


Object
MarshalByRefObject
Component
Control
ScrollableControl
ContainerControl
Form

Imenski prostor

- Imenski prostor (*namespace*) je kontekst u kojem postoje različiti identifikatori.
- Identifikator definiran u imenskom prostoru je povezan s tim imenskim prostorom.
- Isti identifikator može neovisno postojati u različitim imenskim prostorima.
- Identifikatori u programskom kodu su imena varijabli, klasa, objekata, funkcija,.....

Izvor: www.wikipedia.org

Imenski prostor

- Programski jezici koji podržavaju korištenje imenskih prostora definiraju pravila po kojima određuju kojem imenskom prostoru identifikator pripada.
- Kod veliki računalnih programa nije neuobičajeno postojanje na tisuće identifikatora. Imenski prostor pruža mehanizam skrivanja lokalnih identifikatora. Na taj se način mogu grupirati identifikatori u odgovarajuće imenske prostore i na taj je način sustav više modularan.

Imenski prostor

- Većina novijih programskih jezika ima podršku za imenske prostore.
- Kod nekih programskih jezika poput C++ ili Pythona identifikatori koji označavaju određeni imenski prostor su i sami ograničeni drugim imenskim prostorom, pa se imenski prostori mogu gnijezditi i stvarati stablo imenskog prostora (*namespace tree*).
- Osnovni element na vrhu stabla (*root*) je neimenovani globalni imenski prostor (*global namespace*).

C++ kod

```
#include <iostream>
using namespace std;
void main()
{
    cout<<“C++ program”<<endl;
}
```

Definiranje vlastitog imenskog prostora:

```
namespace moj
{
    int a;
}
```

Doseg idenfitikatora

- Različiti programski jezici imaju različite tipove dosega (*scope*) idenitifikatora.
- Tip dosega određuje što sve može biti navedeno u određenom dosegu i na koji način.
- Ovisno o tipu dosega, doseg može:
  - sadržavati deklaracije ili definicije identifikatora
  - sadržavati naredbe ili izraze koji definiraju izvršni algoritam ili jedan njegov dio
  - biti ugniježđen ili sadržavati druge ugniježđene dosege.

Doseg idenfitikatora

- Imenski prostor je ustvari jedan tip dosega koji koristi ograničavajuća tj. omeđujuća svojstva dosega da grupira lokalno povezane identifikatore pod jedan identifikator (ime imenskog prostora).
- Glavni razlog korištenja dosega je na neki način odvajanje i razlikovanje varijabli u različitim dijelovima programa.

Različiti tipovi dosega u C++ kodu

```
namespace N
{ // namespace scope, merely groups identifiers
    class C
    { // class scope, defines/declares member variables and functions
        void f (bool b)
        { // function scope, contains executable statements
            if (b) {} // unnamed scope for conditionally executed
                      // statements ...
        }
    };
}
```

.NET imenski prostor

- Kod .NET-a imenski prostor služi primarno za organiziranje objekata u assemblyju. Assembly može sadržavati jedan ili više imenskih prostora koji opet mogu sadržavati ugniježđene imenske prostore.
- Imenski prostor sprječava nejasnosti kada se koristi velika skupina objekata poput biblioteke klasa u .NET-u i organizira reference. Za razliku od C++ kojem je root imenski prostor globalni imenski prostor i nema neki posebni identifikator root imenski prostor za NGWS (.NET je početno zvan NGWS - *Next Generation Windows* (ili Web) *Service*s) je *System*.

.NET imenski prostor

- Unutar root imenskog prostora se nalaze npr. klase koje definiraju osnovne tipove podataka. Pored toga tu se nalaze i klase za rad s *garbage collector-om*, za rad s iznimkama (*exception handling*) i sl.
- Osim toga root imenski prostor na sljedećoj nižoj razini (organizacija je hijerarhijska) uključuje druge imenske prostore (otprilike njih 24).

.NET imenski prostor

Data	System.Data System.XML
Component Model	System.CodeDOM System.ComponentModel System.Core
Configuration	System.Configuration
Frameworks Services	System.Diagnostics System.DirectoryServices System.Timers System.Messaging System.ServiceProcess
Globalization	System.Globalization System.Resources
Net	System.Net

.NET imenski prostor

Programming Basics	System.Collection System.IO System.Text System.Threading
Reflection	System.Reflection
Rich Client-Side GUI	System.Drawing System.Winforms
Runtime Infrastructure Services	System.Runtime
Security	System.Security
Web Services	System.Web

C# projekt

- Visual Studio prilikom generiranja novog C# projekta odmah kreira imenski prostor u koji se smještaju sve klase aplikacije.
- Naredba *namespace* kaže da sve što se navede nakon te naredbe i identifikatora imenskog prostora unutar {} zagrada je dio upravo tog imenskog prostora.

C# projekt

- Kada poslije u kodu definiramo klasu npr. public class Form1 : System.Windows.Forms.Form pravo ime te klase nije Form1 nego ime_imenskog_prostora.Form1 i ukoliko kôd izvan tog imenskog prostora koristi tu klasu mora je dohvaćati preko punog imena ili tzv. *fully-qualified name*.
- Klasa Form1 je izvedena iz klase Form koja je definirana u imenskom prostoru Windows.Forms, koji je definiran u imenskom prostoru System.

C# projekt

- Za korištenje određenog imenskog prostora u projektu koristi se naredba *using*. Pomoću te naredbe je omogućeno izbjegavanje pisanja *fully-qualified name* klase.
- Npr:

  using System.Windows.Forms;

  public class Form1 : System.Windows.Forms.Form =

  public class Form1 : Form

C# projekt

- Korištenje *using* naredbe kaže ustvari kompajleru u kojim imenskim prostorima da traži klase koje se pozivaju.
- Ukoliko slučajno klasa s istim imenom postoji u dva imenska prostora kompajler će generirati grešku i onda se za pristupanje klase mora koristiti *fully-qualified name.*