



OBJEKTNO ORIJENTISANO PROJEKTOVANJE

WINDOWS PROGRAMIRANJE MICROSOFT VISUAL C#

1

Katedra za računarstvo
Elektronski fakultet u Nišu

SADRŽAJ

- Kratak osvrt na jezik *Microsoft Visual C#*
- Uvod u vizualno programiranje u *Microsoft Visual C#*



MS VISUAL C#

- Svojstva (**property**)
- Parcijalne klase
- Anonimni tip podataka (**var**)
- Metode proširenja (**Extension methods**)
- Operator **is** i **as**
- Delegati (**delegate**)
- Događji (**event**)
- **Value type i Reference type**
- **Nullabe type**
- **Boxing i Unboxing**
- **Prenos parametara metodi** (po vrednosti, referenci) u zavisnosti da li je podatak koji se prenosi funkciji vrednosnog ili referencnog tipa

SVOJSTVA (PROPERTY)

```
class Ucenik
{
    public string Ime;
    public string Prezime;
    private int prRazred;
    public int Razred
    {
        get
        {
            return prRazred;
        }
        set
        {
            prRazred = value;
        }
    }
}
```

```
public int Razred
{
    get
    {
        return prRazred;
    }
    set
    {
        if (value > 0 && value < 5)
        {
            prRazred = value;
        }
        else
        {
            // greska
        }
    }
}
```

PARCIJALNE KLASSE

- Mogućnost da kreirate klase u više fajlova.
- Koristi se ključna reč **partial**
 - Npr. u jednom fajlu mogu da se nalaze metode, u drugom svojstva, u trećem atributi, ...

```
public partial class CoOrds
{
    private int x;
    private int y;

    public CoOrds(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```

```
public partial class CoOrds
{
    public void PrintCoOrds()
    {
        Console.WriteLine("CoOrds: {0},{1}", x, y);
    }
}
```

```
CoOrds myCoOrds = new CoOrds(10, 15);
myCoOrds.PrintCoOrds();
```

ANONIMNI TIP PODATAKA (**var**)

```
var v = new { NekiBroj= 108, NekiTekst = "HelloWorld!" };  
  
Console.WriteLine("Broj: " + v.NekiBroj);  
Console.WriteLine("NekiTekst: " + v.NekiTekst);
```

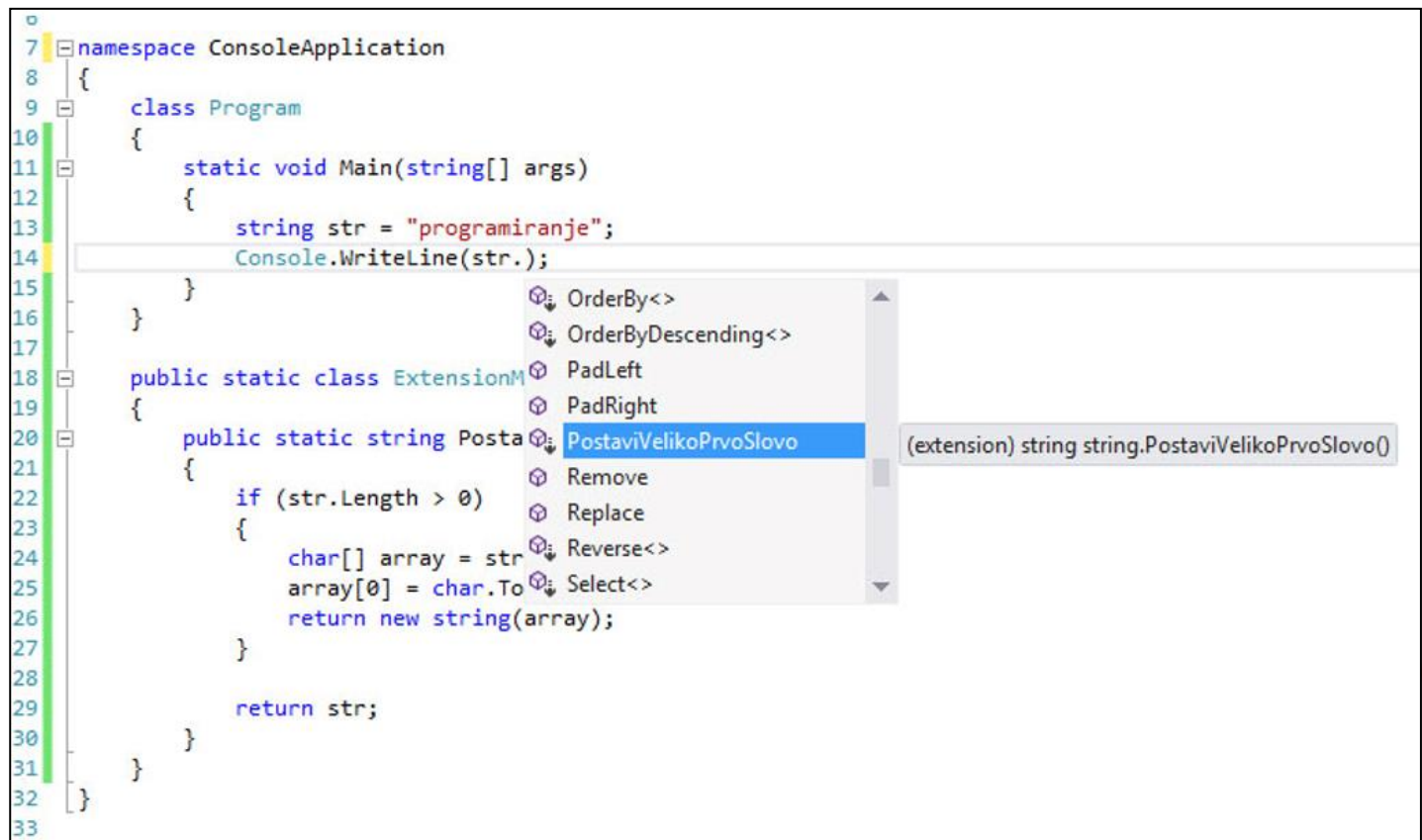
```
List<Automobil> listaAutomobila = new List<Automobil>();  
  
listaAutomobila.Add(new Automobil());  
listaAutomobila.Add(new Automobil());  
listaAutomobila.Add(new Automobil());  
listaAutomobila.Add(new Automobil());  
listaAutomobila.Add(new Automobil());  
  
foreach (var a in listaAutomobila)  
    Console.WriteLine(a.model);
```

foreach petlja

METODE PROŠIRENJA (EXTENSION METHODES)

```
public static class ExtensionMethods
{
    public static String PostaviVelikoPrvoSlovo(this String str)
    {
        if (str.Length > 0)
        {
            char[] array = str.ToCharArray();
            array[0] = char.ToUpper(array[0]);
            return new String(array);
        }

        return str;
    }
}
```



OPERATOR **IS** I **AS**

- Operator **is** ima vrednost **true** i **false**.
 - Koristi se za proveru da li je tip objekta onaj koji mislite da jeste.
 - `if(o1 is Krug) {...}`
- Operator **as**

```
class Base
{
    public override string ToString()
    {
        return "Base";
    }
}
class Derived : Base
{ }
```

```
class Program
{
    static void Main()
    {
        Derived d = new Derived();
        Base b = d as Base;
        if (b != null)
        {
            Console.WriteLine(b.ToString());
        }
    }
}
```


DELEGATI (delegate)

- Delegat je referencijalni tip koji se može koristiti za enkapsulaciju imenovane ili anonimne metode.
- Delegati su osnova za kreiranje događaja (eng. *Events*).
- Na .NET platformi ne postoje pokazivači na funkcije a njihova zamena su delegati (npr. pokazivači na funkcije C, C++).
- Upotrebom delegata možete posredno pozivati jednu ili više funkcija.
- Npr. funkcija za sortiranje niza. Obično takve funkcije zahtevaju dva argumenta – prvi je niz koji treba sortirati a drugi bi bila funkcija za sortiranje koja će određivati da li je jedan element niza veći/manji od drugog po nekom kriterijumu.

DELEGATI (delegate)

```
[modifikatori] delegate povratni-tip ime-delegata ([parametri]);
```

- Delegati mogu pozivati više od jedne funkcije. Sve funkcije koje će određeni delegat pozivati sačinjavaju “**listu poziva**” tog delegata. Za dodavanje neke funkcije toj listi koristi se operator **+=**, a za uklanjanje neke od funkcija iz te liste koristi se operator **-=**. Delegati se instanciraju dodeljivanjem imenovane ili anonimne metode.
- Upotrebom anonimne metode nije potrebno prethodno definisati metode koje se pridružuju delegatu (dodaju u listu poziva tog delegata) u telu neke klase. Moguće je jednostavno vezati anonimnu metodu za delegat odmah pored instanciranja delegata (Stampac s = **delegate**(string j)). Kreiranje anonimne metode je osnovni način za prosleđivanje koda koji je vezan za delegat i izvršiće se onda kada sa pozove delegat negde u kodu.

```
delegate void Stampac(string s);  
  
// instanciranje delegate s anonimnom metodom Stampaj  
Stampac s = delegate(string j)  
{  
    Console.WriteLine(j);  
};
```

```
delegate void Stampac(string s);  
  
void Stampaj(string tekstZaStampu)  
{  
    Console.WriteLine(j);  
};  
  
// instanciranje delegata s imenovanom metodom Stampaj  
Stampac s = obj.Stampaj("Neki tekst za stampanje.");
```

DELEGATI (delegate)

- Potpisu delegata (povratni tip, tipovi parametara) moraju da odgovaraju sve metode koje želimo da dodamo u listi poziva delegata.
- Delegati se mogu instancirati pomoću metode ili **lambda izraza** (eng. lambda expression, oznaka lambda izraza je **=>**).
- Primer upotrebe lambda izraza:

```
delegate double Racunaj(double num);  
Racunaj r = s => s * s * s;  
// sintaksa za prikaz lambda expression-a je =>  
  
double rez = r(5);  
  
Console.WriteLine("Rezultat: " + r);  
// Rezultat je da promenljiva rez sadzi vrednost 125
```

DOGAĐAJI (*events*)

- Upotrebom događaja moguće je obavestiti korisnika da se nešto desilo sa objektom kako bi korisnik preduzeo neku akciju.
- Programer može da definiše parče koda koje se izvršava kada se neki događaj desi.
- Najčešće se događaji koriste kod aplikacija sa grafički korisničkim interfejsom (*eng. GUI – Graphical User Interface*).
- Događaji omogućavaju klasama ili objektima da obavestave druge klase ili objekte kada se nešto od nekog interesa desilo.
- Klasa koja okida događaj se zove generator događaja (*eng. **publisher***) a klasa koja obrađuje (hendluje, *eng. **handle***) događaj naziva se pretplatnik (*eng. **subscriber***).
- Generator događaja određuje kada će neki događaj da se desi.
- Pretplatnik određuje koja će se akcija izvršiti kao odgovor na generisani događaj.
- Događaj može da ima više pretplatnika. Pretplatnik može da opsluži više generatora događaja. Događaj koji nema pretplatnika nikada se ne podiže. U .NET-u događaji su bazirani na delegat **EventHandler** i osnovnu klasu **EventArgs**.

```
[modifikatori] event ime-klase-delegata ime-događaja;
```

DOGAĐAJI (*events*) - PRIMERI

- Kada kliknete na neko dugme (*eng. button*) na formi (*eng. form*) aktivira će se događaj *Click* a kao rezultat tog događaja desiće se neka akcija (npr. pojaviće se *MessageBox* sa nekom porukom). Aplikacija detektuje da je korisnik pritisnuo dugme, to dugme na koje je korisnik kliknuo podiže događaj *Click* a taj događaj izvršava neko parče koda koje je programer napisao (npr. poziva se *MessageBox* sa nekom tekstualnom porukom).
- Prilikom zatvaranja forme može se izvršiti deo koda koji je dodeljen događaju *Closing* i pitati korisnika preko dijaloga da li je siguran sa akcijom zatvaranja forme.
- Visual Studio automatski, nakon klika na *button*, kreira metodu koja obrađuje događaj *Click* i automatski je dodeljuje tom događaju.
- Na panelu za svojstva (*eng. Properties panel*) koji se odnosi na događaje moguće je videti sve događaje koje podržava neka .NET kontrola nakon što je ona odabrana. Klikom na neki događaj iz liste, automatski se generiše metoda koja opslužuje (hendluje) izabrani događaj. Programeru ostaje samo da popuni tu metodu potrebnim kodom. Naravno moguće je direktno u kodu hendlovati neki događaj bez posezanjem za panelom svojstva. Upotreba događaja nije svedena samo na aplikacije sa grafičko korisničkim interfejsom. Događaji omogućavaju da se izvrši neki kod kada se neki događaj desi. Zbog toga događaji su usko vezani sa delegatima. Za kreiranje događaja koristi se ključna reč *event*.

DOGAĐAJI (events) - PRIMERI

```
using System;

namespace ConsoleApplicationAutomobileEvents
{
    class Program
    {
        public class Automobil
        {
            public delegate void PrazanRezervoarHandler();
            public event PrazanRezervoarHandler OnPrazanRezervoar;

            public delegate void NaRezerviHandler();
            public event NaRezerviHandler OnNaRezervi;

            public event EventHandler OnNapunjenRezervoar;

            private const int _kapacitetRezervoara = 50; // u litrima
            private int _litraURezervoaru = 0;

            public int OstaloBenzina
            {
                get { return _litraURezervoaru; }
                set { _litraURezervoaru = value; }
            }

            public Automobil()
            {
                // inicijalano postavljamo da je rezervoar automobila pun
                _litraURezervoaru = _kapacitetRezervoara;
            }
        }
    }
}
```

DOGAĐAJI (events) - PRIMERI

```
public void VoziAutomobil()
{
    _litraURezervoaru -= 10;

    if (_litraURezervoaru < 0)
        _litraURezervoaru = 0;

    if (_litraURezervoaru < 15 && _litraURezervoaru > 0)
    {
        if (OnNaRezervi != null)
            OnNaRezervi();
    }
    else if (_litraURezervoaru == 0)
    {
        if (OnPrazanRezervoar != null)
            OnPrazanRezervoar();
    }
}

public void NapuniRezervoar()
{
    _litraURezervoaru = _kapacitetRezervoara;

    if (OnNapunjenRezervoar != null)
        OnNapunjenRezervoar(this, null);
}
```

DOGAĐAJI (events) - PRIMERI

```
public void SipajGorivo(int litara)
{
    _litraURezervoaru += litara;

    if (_litraURezervoaru > _kapacitetRezervoara)
        _litraURezervoaru = _kapacitetRezervoara;

    if (OnNapunjenRezervoar != null)
        OnNapunjenRezervoar(this, null);
}

static void Main(string[] args)
{
    Automobil golf = new Automobil();

    golf.OnNaRezervi += golf_OnNaRezerviHandler;
    golf.OnPrazanRezervoar += golf_OnPrazanRezervoar;
    golf.OnNapunjenRezervoar += golf_OnNapunjenRezervoarHandler;

    golf.VoziAutomobil();
    golf.VoziAutomobil();
    golf.VoziAutomobil();
    golf.VoziAutomobil();
    golf.SipajGorivo(20);
    golf.VoziAutomobil();
    golf.VoziAutomobil();
    golf.NapuniRezervoar();
    golf.VoziAutomobil();
    golf.VoziAutomobil();
    golf.VoziAutomobil();
    golf.VoziAutomobil();
    golf.VoziAutomobil();
}
```


DOGAĐAJI (events) - PRIMERI

```
static void golf_OnNapunjenRezervoarHandler(Object sender,
EventArgs e)
{
    int litaraURezervoaru = (sender as Automobil).OstaloBenzina;
    Console.WriteLine("Sipali ste gorivo. Sada imate "
        + litaraURezervoaru
        + " l u rezervoaru.");
}

static void golf_OnPrazanRezervoar()
{
    Console.WriteLine("Nemate više goriva u rezervoaru. Sacekajte
pomoc!");
}

static void golf_OnNaRezerviHandler()
{
    Console.WriteLine("Gorivo je na rezervi. Sipajte gorivo prvom
prilikom!");
}
}
```

PRENOS PARAMETARA FUNKCIJI

○ Vrednosni tipovi

- Prenos vrednosnog tipa po vrednosti
- Prenos vrednosnog tipa po referenci

○ Referentni tipovi

- Prenos referentnog tipa po vrednosti
- Prenos referentnog tipa po referenci

PRENOS VREDNOSNOG TIPA PO VREDNOSTI

```
class PassingValByVal
{
    static void SquareIt(int x)
    // The parameter x is passed by value.
    // Changes to x will not affect the original value of x.
    {
        x *= x;
        System.Console.WriteLine("The value inside the method: {0}", x);
    }
    static void Main()
    {
        int n = 5;
        System.Console.WriteLine("The value before calling the method: {0}", n);

        SquareIt(n); // Passing the variable by value.
        System.Console.WriteLine("The value after calling the method: {0}", n);

        // Keep the console window open in debug mode.
        System.Console.WriteLine("Press any key to exit.");
        System.Console.ReadKey();
    }
}
/* Output:
    The value before calling the method: 5
    The value inside the method: 25
    The value after calling the method: 5
*/
```

PRENOS VREDNOSNOG TIPa PO REFERENCI

```
class PassingValByRef
{
    static void SquareIt(ref int x)
    // The parameter x is passed by reference.
    // Changes to x will affect the original value of x.
    {
        x *= x;
        System.Console.WriteLine("The value inside the method: {0}", x);
    }
    static void Main()
    {
        int n = 5;
        System.Console.WriteLine("The value before calling the method: {0}", n);

        SquareIt(ref n); // Passing the variable by reference.
        System.Console.WriteLine("The value after calling the method: {0}", n);

        // Keep the console window open in debug mode.
        System.Console.WriteLine("Press any key to exit.");
        System.Console.ReadKey();
    }
}
/* Output:
    The value before calling the method: 5
    The value inside the method: 25
    The value after calling the method: 25
*/
```

ZAMENA DVE VREDNOSTI

```
static void SwapByRef(ref int x, ref int y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

```
static void Main()
{
    int i = 2, j = 3;
    System.Console.WriteLine("i = {0}  j = {1}" , i, j);

    SwapByRef (ref i, ref j);

    System.Console.WriteLine("i = {0}  j = {1}" , i, j);

    // Keep the console window open in debug mode.
    System.Console.WriteLine("Press any key to exit.");
    System.Console.ReadKey();
}
/* Output:
    i = 2  j = 3
    i = 3  j = 2
*/
```

PRENOS REFERENTNOG TIPa PO VREDNOSTI

```
class PassingRefByVal
{
    static void Change(int[] pArray)
    {
        pArray[0] = 888; // This change affects the original element.
        pArray = new int[5] {-3, -1, -2, -3, -4}; // This change is local.
        System.Console.WriteLine("Inside the method, the first element is: {0}", pArray[0]);
    }

    static void Main()
    {
        int[] arr = {1, 4, 5};
        System.Console.WriteLine("Inside Main, before calling the method, the first element is: {0}", arr [0]);

        Change(arr);
        System.Console.WriteLine("Inside Main, after calling the method, the first element is: {0}", arr [0]);
    }
}

/* Output:
    Inside Main, before calling the method, the first element is: 1
    Inside the method, the first element is: -3
    Inside Main, after calling the method, the first element is: 888
*/
```

PRENOS REFERENTNOG TIPA PO REFERENCI

```
class PassingRefByRef
{
    static void Change(ref int[] pArray)
    {
        // Both of the following changes will affect the original variables:
        pArray[0] = 888;
        pArray = new int[5] {-3, -1, -2, -3, -4};
        System.Console.WriteLine("Inside the method, the first element is: {0}", pArray[0]);
    }

    static void Main()
    {
        int[] arr = {1, 4, 5};
        System.Console.WriteLine("Inside Main, before calling the method, the first element is: {0}", arr[0]);

        Change(ref arr);
        System.Console.WriteLine("Inside Main, after calling the method, the first element is: {0}", arr[0]);
    }
}

/* Output:
    Inside Main, before calling the method, the first element is: 1
    Inside the method, the first element is: -3
    Inside Main, after calling the method, the first element is: -3
*/
```

ZAMENA VREDNOSTI DVA STRINGA

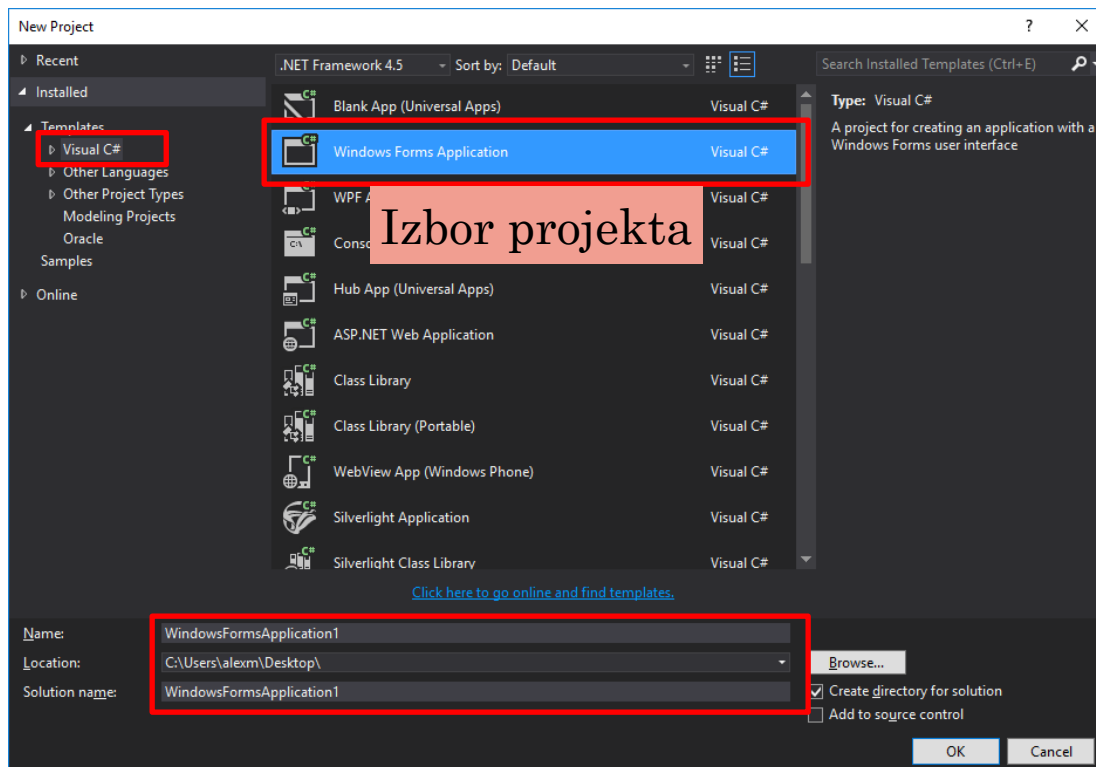
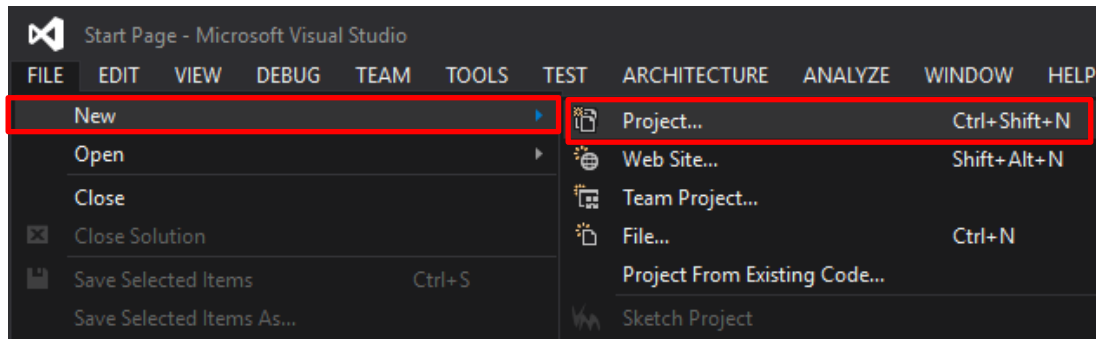
```
class SwappingStrings
{
    static void SwapStrings(ref string s1, ref string s2)
    // The string parameter is passed by reference.
    // Any changes on parameters will affect the original variables.
    {
        string temp = s1;
        s1 = s2;
        s2 = temp;
        System.Console.WriteLine("Inside the method: {0} {1}", s1, s2);
    }

    static void Main()
    {
        string str1 = "John";
        string str2 = "Smith";
        System.Console.WriteLine("Inside Main, before swapping: {0} {1}", str1, str2);

        SwapStrings(ref str1, ref str2);    // Passing strings by reference
        System.Console.WriteLine("Inside Main, after swapping: {0} {1}", str1, str2);
    }
}

/* Output:
    Inside Main, before swapping: John Smith
    Inside the method: Smith John
    Inside Main, after swapping: Smith John
*/
```


KREIRANJE PROJEKTA U VISUALSTUDIO



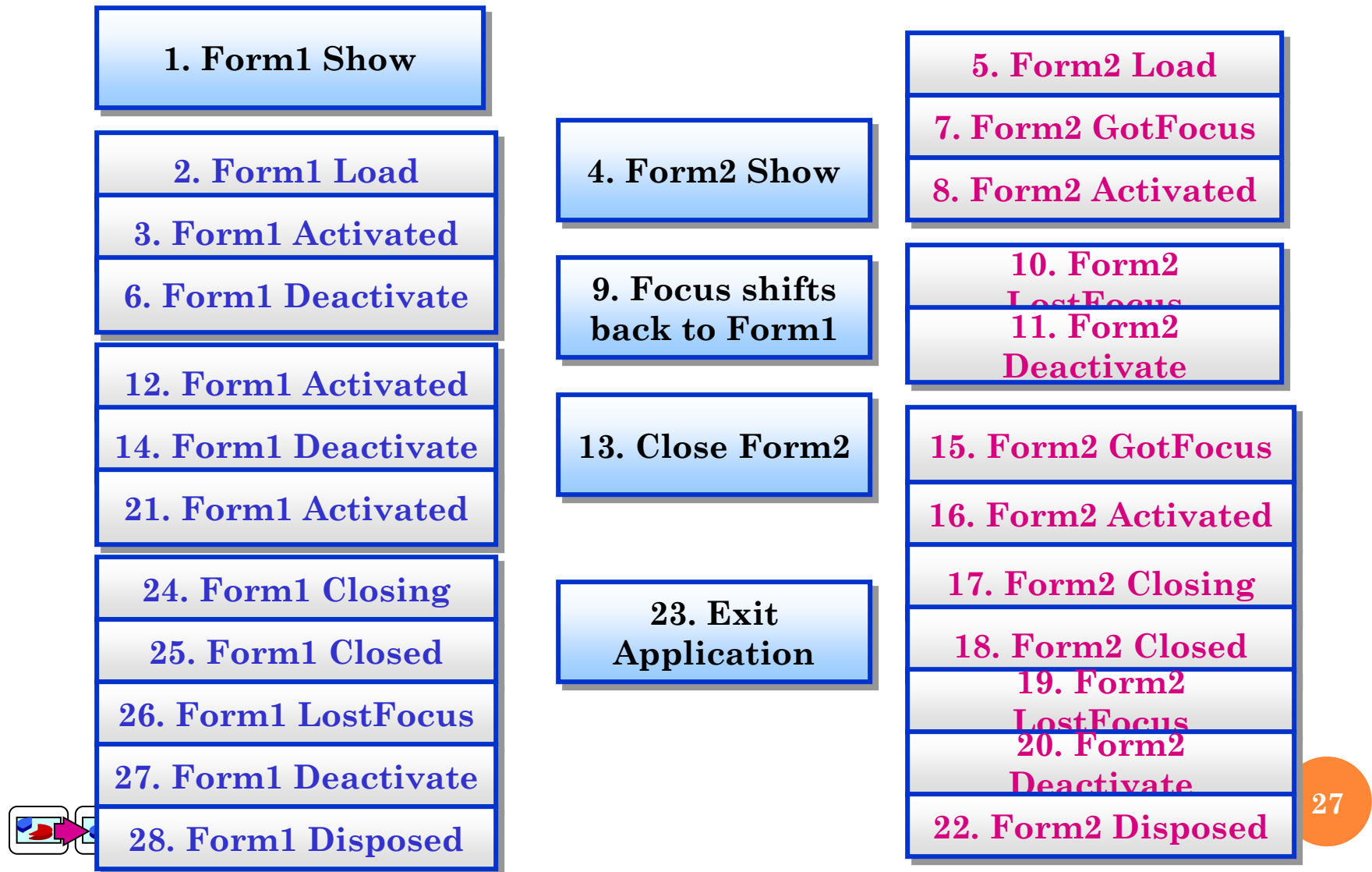
NOVI PROJEKAT

The screenshot displays the Microsoft Visual Studio environment for a new project named 'WindowsFormsApplication1'. The interface is divided into several panes:

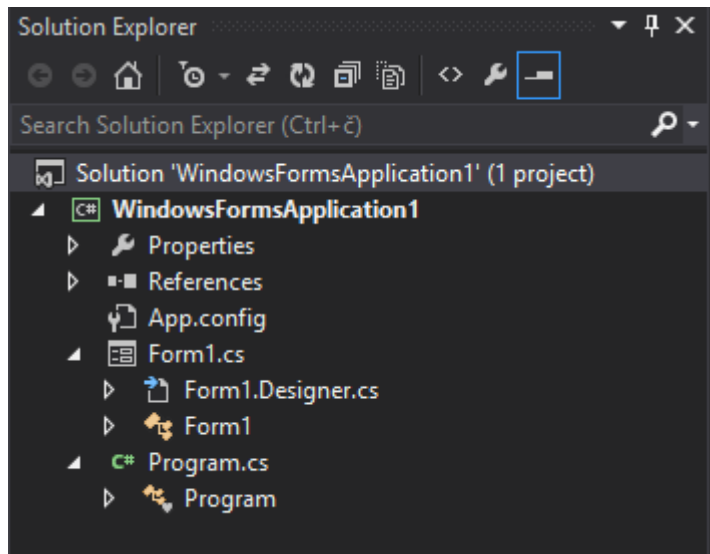
- Toolbox:** Located on the left, it contains a list of 'Common Controls' such as Pointer, Button, CheckBox, and ListBox. A red box highlights this list, with the annotation 'Lista kontrola Common controls'.
- Form1.cs [Design]:** The central pane shows a design view of a new form named 'Form1'. A green box highlights this area, with the annotation 'Kreirana forma Form1'.
- Solution Explorer:** Located on the right, it shows the project structure. A blue box highlights the 'Solution Explorer' pane, with the annotation 'Solution Explorer'.
- Properties:** Below the Solution Explorer, the Properties window shows the properties of the selected 'Form1' object. A yellow box highlights this window, with the annotation 'Properties'.

The bottom status bar indicates the application is 'Ready'.

ŽIVOTNI CIKLUS FORME



SOLUTION EXPLORER



Jedan **Solution** može da sadrži više projekta

- Solution '**WindowsFormsApplication1**'
 - Projekat **WindowsFormsApplication1**

Prilikom kreiranja novog projekta kreira se nova forma: **Form1** i fajl **Program.cs**.

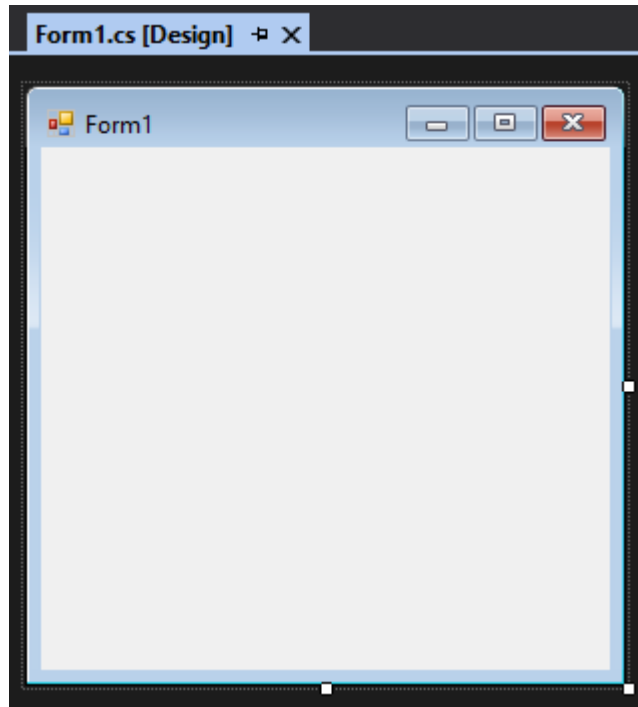
Pri tome se kreiraju dva fajla koja opisuju **Form**

1. **Form1.Designer.cs**
2. **Form1.cs**

Ukoliko nije prikazan Solution Explorer potrebno je ići na meni **View/Solution Explorer**

This PC > Desktop > WindowsFormsApplication1 > WindowsFormsApplication1				
Name	Date modified	Type	Size	
bin	10/10/2016 14:27	File folder		
obj	10/10/2016 14:27	File folder		
Properties	10/10/2016 14:27	File folder		
App.config	10/10/2016 14:27	XML Configuratio...	1 KB	
Form1.cs	10/10/2016 14:27	CS File	1 KB	
Form1.Designer.cs	10/10/2016 14:27	CS File	2 KB	
Program.cs	10/10/2016 14:27	CS File	1 KB	
WindowsFormsApplication1.csproj	10/10/2016 14:27	Visual C# Project f...	4 KB	

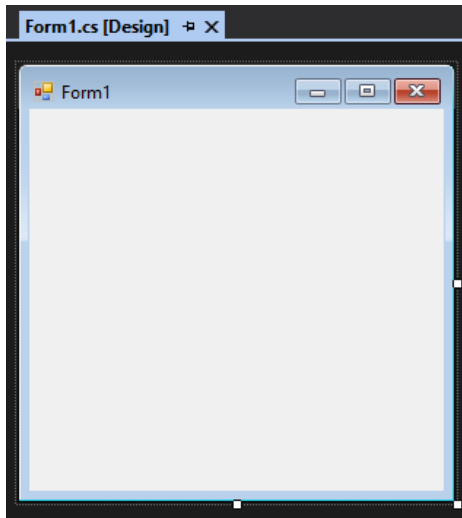
FORM1.CS [DESIGN], FORM1.CS



A screenshot of the Visual Studio Code view for the same 'Form1.cs' file. The window title bar shows 'Form1.cs' and 'Form1.cs [Design]'. The code is written in C# and includes several using statements and a partial class definition for 'Form1'.

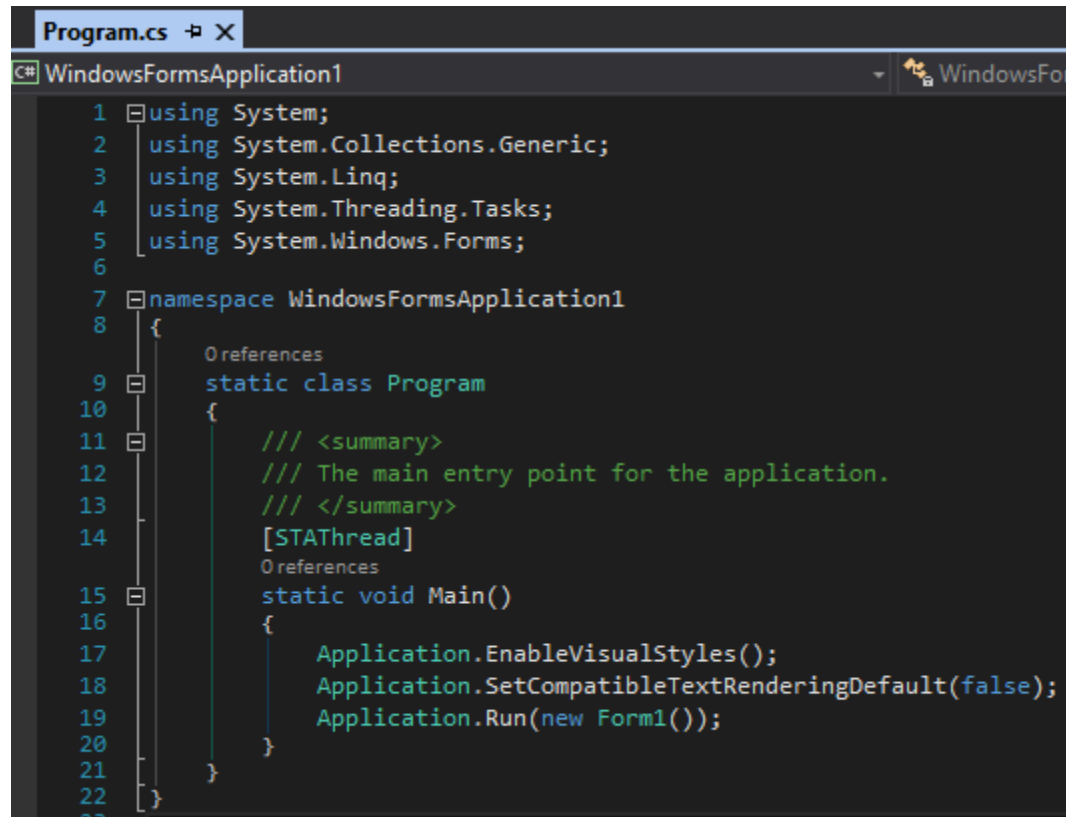
```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10
11 namespace WindowsFormsApplication1
12 {
13     3 references
14     public partial class Form1 : Form
15     {
16         1 reference
17         public Form1()
18         {
19             InitializeComponent();
20         }
21     }
```

FORM1.CS [DESIGN], FORM1.DESIGNER.CS



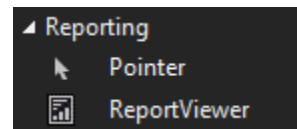
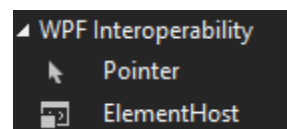
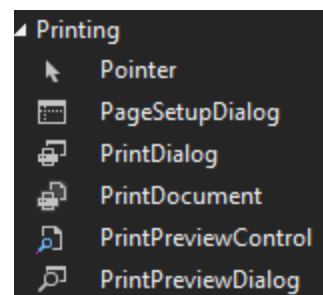
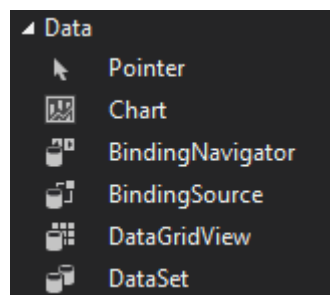
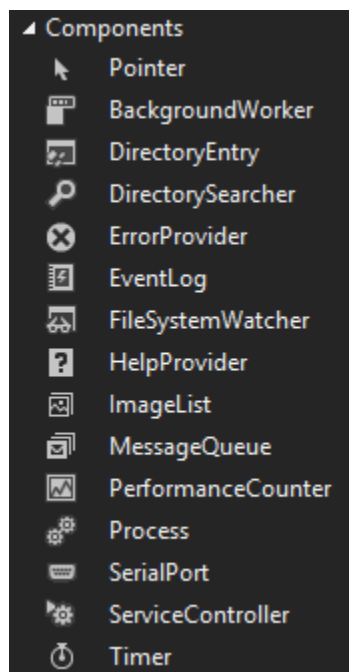
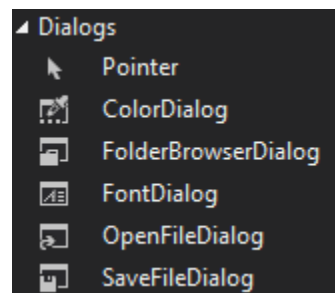
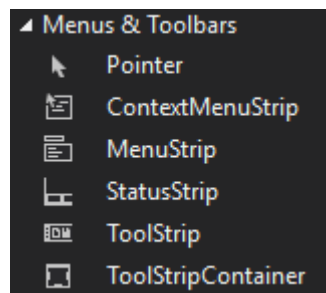
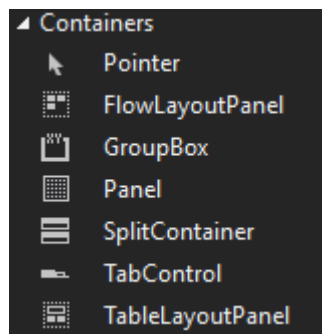
```
Form1.cs  Form1.cs [Design]  Form1.Designer.cs  X
WindowsFormsApplication1
1 namespace WindowsFormsApplication1
2 {
3     3 references
4     partial class Form1
5     {
6         /// <summary>
7         /// Required designer variable.
8         /// </summary>
9         private System.ComponentModel.IContainer components = null;
10
11        /// <summary>
12        /// Clean up any resources being used.
13        /// </summary>
14        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
15        1 reference
16        protected override void Dispose(bool disposing)
17        {
18            if (disposing && (components != null))
19            {
20                components.Dispose();
21            }
22            base.Dispose(disposing);
23        }
24
25        #region Windows Form Designer generated code
26
27        /// <summary>
28        /// Required method for Designer support - do not modify
29        /// the contents of this method with the code editor.
30        /// </summary>
31        1 reference
32        private void InitializeComponent()
33        {
34            this.components = new System.ComponentModel.Container();
35            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
36            this.Text = "Form1";
37        }
38
39        #endregion
40    }
```

PROGRAM.CS

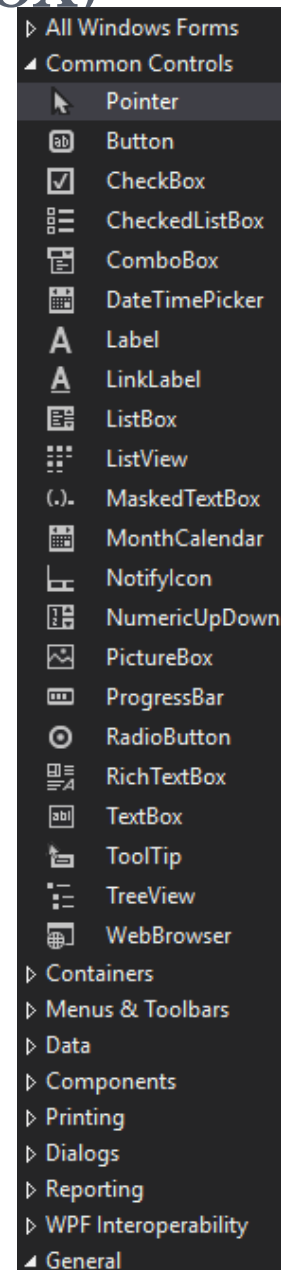


```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using System.Windows.Forms;
6
7 namespace WindowsFormsApplication1
8 {
9     static class Program
10     {
11         /// <summary>
12         /// The main entry point for the application.
13         /// </summary>
14         [STAThread]
15         static void Main()
16         {
17             Application.EnableVisualStyles();
18             Application.SetCompatibleTextRenderingDefault(false);
19             Application.Run(new Form1());
20         }
21     }
22 }
```

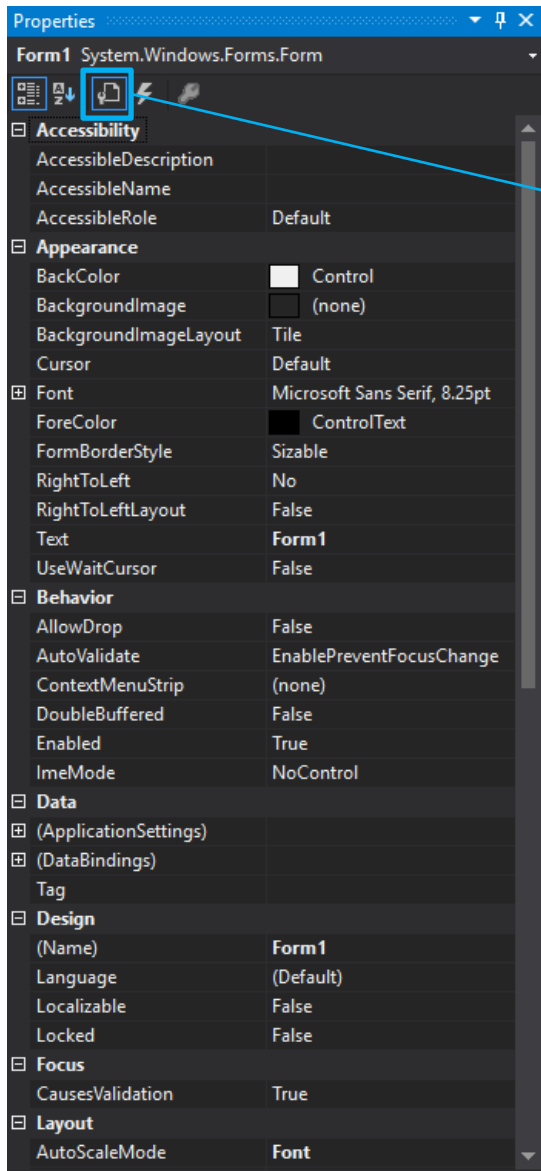
LISTA DOSTUPNIH KONTROLA (TOOLBOX)



Ukoliko nije prikazan panel
Toolbox potrebno je ići na
meni *View/Toolbox*

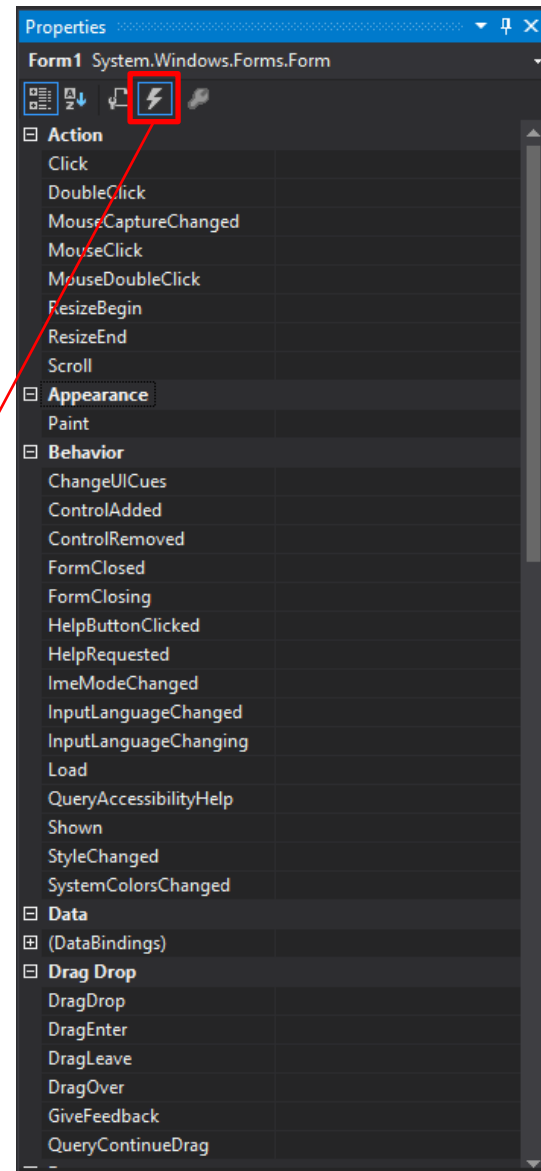
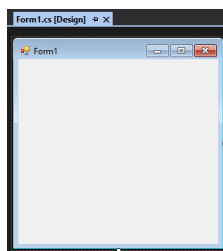


PROPERTIES WINDOW



Lista svojstva (PROPERTIES) koji se odnose za izabranu kontrolu - u ovom slučaju na selektovanu formu **Form1.cs**

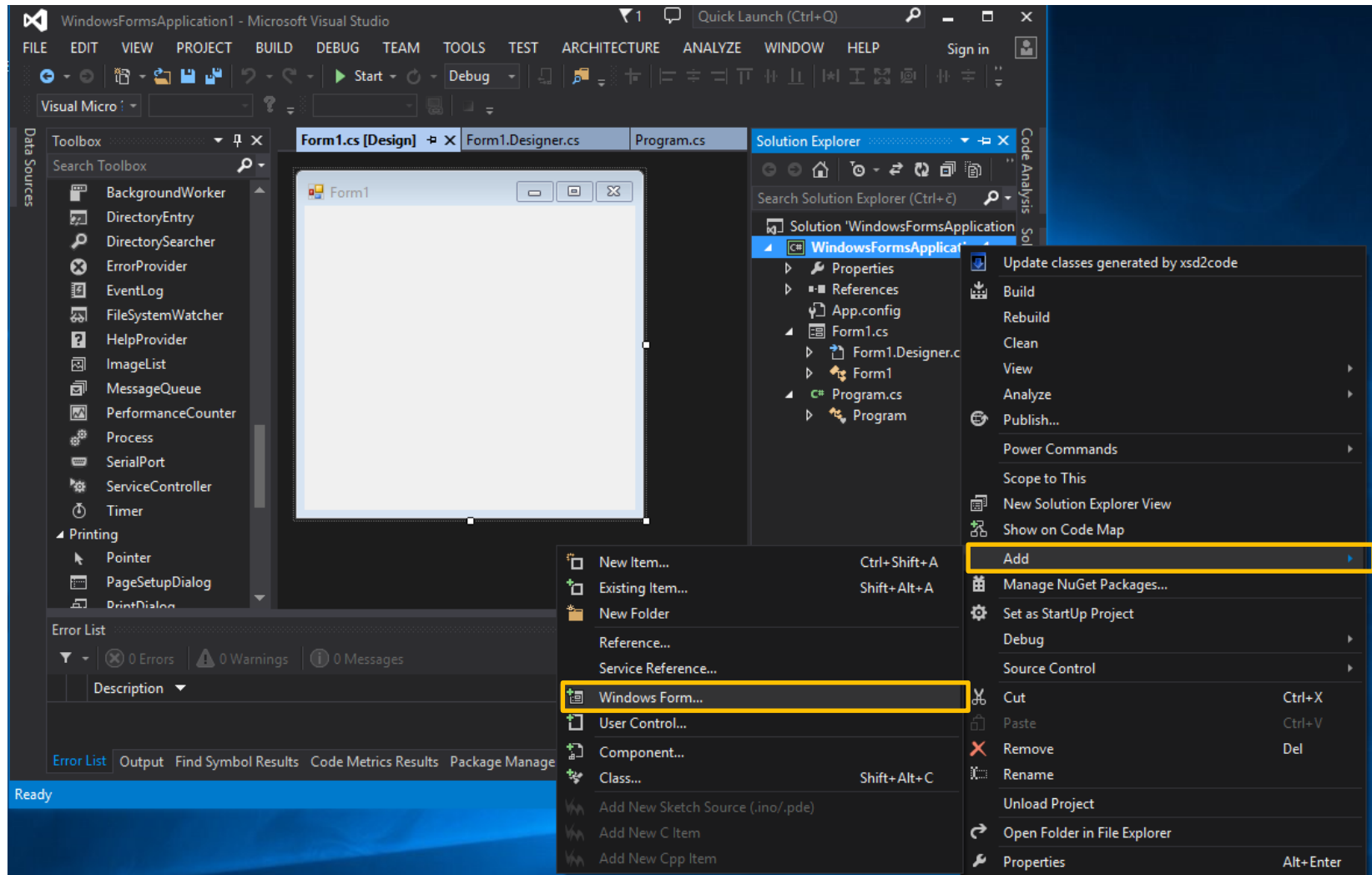
Lista događaja (EVENTS) koji se mogu vezati za izabranu kontrolu - u ovom slučaju na selektovanu formu **Form1.cs**



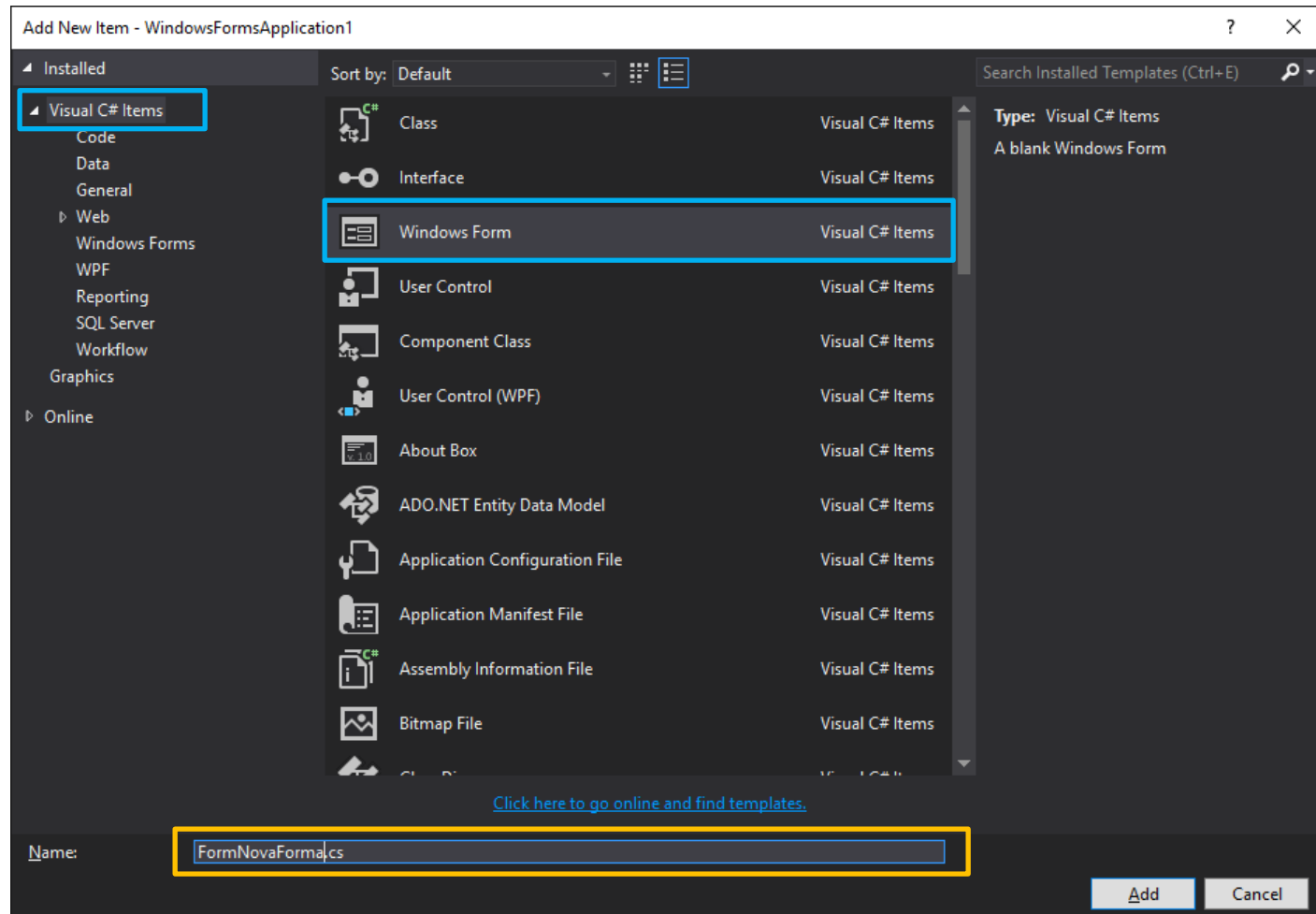
Ukoliko nije prikazan panel **Properties Window** potrebno je ići na meni **View/Properties Window**

DODAVANJE NOVE FORME U POSTOJAĆEM PROJEKTU

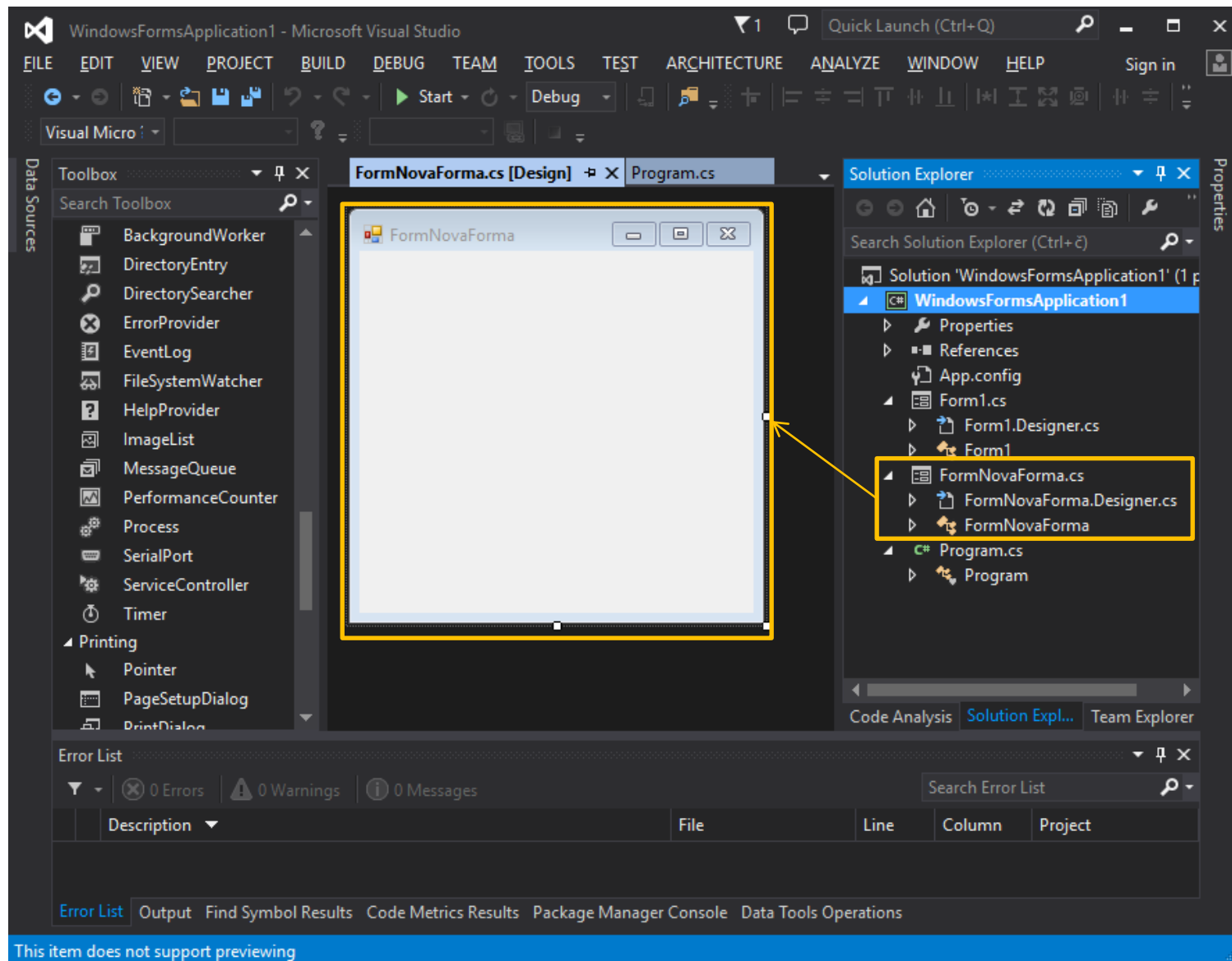
1. Desni klik na projekat
2. Stavka *Add*
3. Stavka *Windows Form ...*



DODAVANJE NOVE FORME U POSTOJAĆEM PROJEKTU

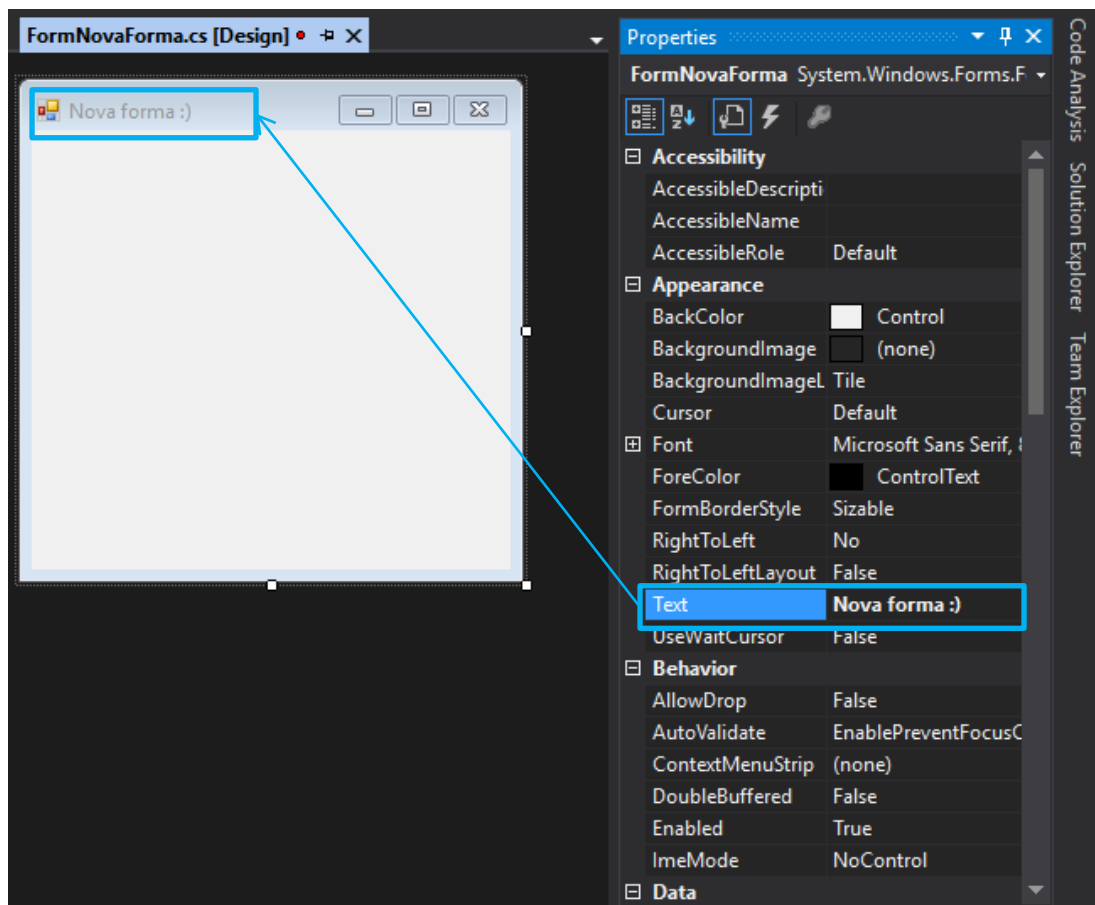


DODAVANJE NOVE FORME U POSTOJAĆEM PROJEKTU



PROMENA NAZIVA FORME

1. Selektovati formu **FormNovaForma**
2. U panelu **Properties Windows** pronaći stavku **Text**
3. Promeniti vrednost stavke **Text** tako da sadrži **Nova forma ☺**



Probajte da promenite
propertyje forme:

*StartPosition, Icon,
Opacity, TopMost,
WindowState, StartPosition*

Sta se dešava?

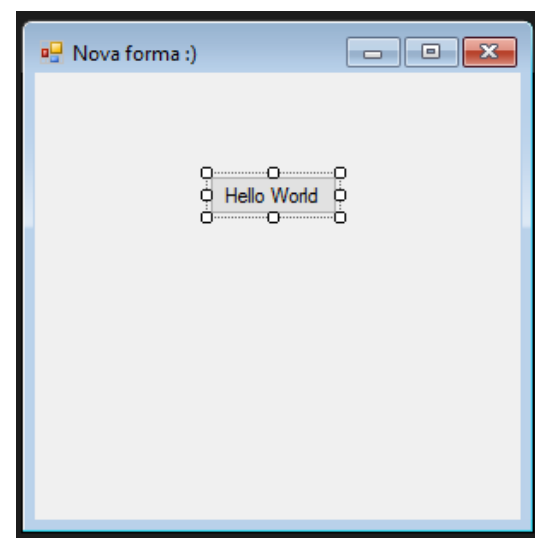
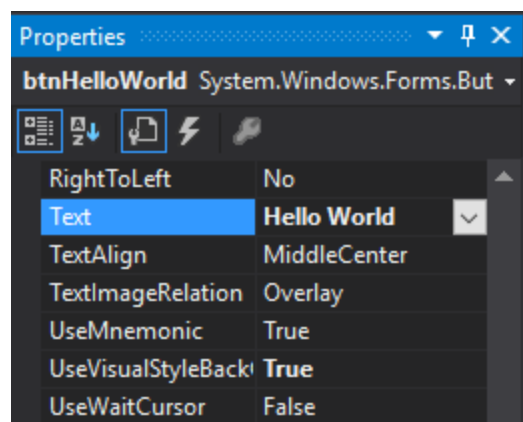
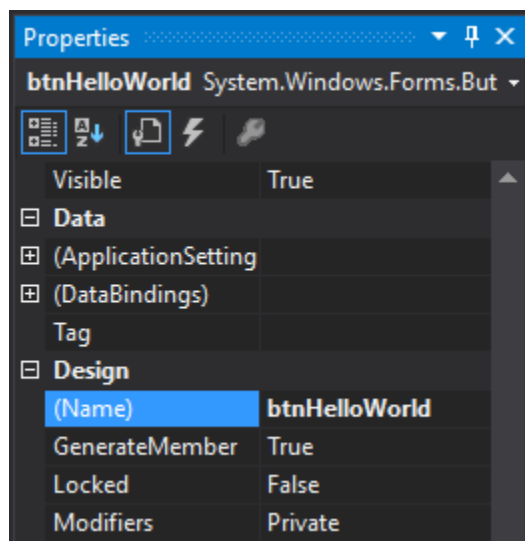
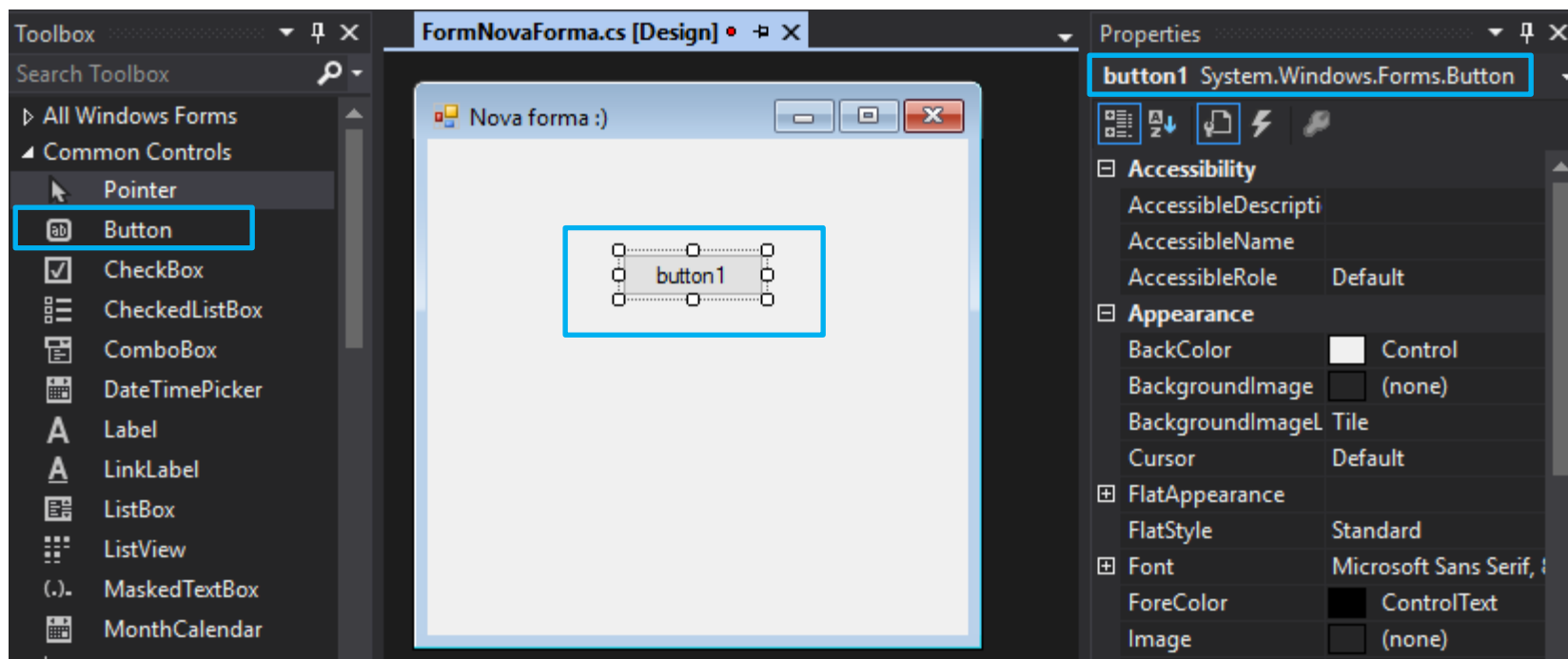
Sta dobijate kao rezultat?

1. Kako možete da
promenite naziv fajla
FormNovaForma.cs?
2. Kako možete da obrišete
formu iz projekta?

DODAVANJE DUGMETA NA FORMU

1. Izabrati **Button** iz panela **Toolbox**
2. Prevući izabranu kontrolu na željenu formu, na željenu poziciju.
3. Promeniti naziv dugmeta, i natpis na dugmetu.
 1. Selektovati dugme.
 2. Promeniti sadržaj svojstva **Text** u panelu **Properties Window**.
 3. Promeniti naziv dugmeta **Button1** sa **btnHelloWorld**.
 1. Promeniti sadržaj svojstva **Name**.
4. Prilikom klika na dugme potrebno je prikazati **MessageBox** sa porukom **Hello World!**
 1. Dva puta kliknuti na dugme.
 2. Upisati sledeći kod `MessageBox.Show("HelloWorld!");`
 - Pokrenite aplikaciju i isprobajte kako sve to radi.
 - Koja se forma pokreće?
 - Kako da pokrenete novokreiranu formu (da ta novokreirana forma bude startna)?
 - Šta se desilo kada ste napravili akciju vezanu za klik na dugme?

DODAVANJE DUGMETA NA FORMU



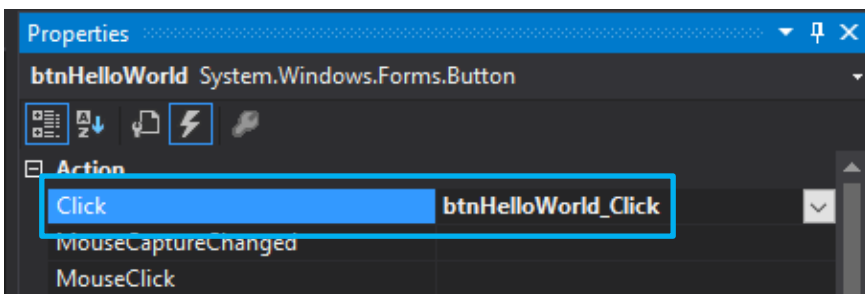
DODAVANJE DUGMETA NA FORMU

Dvostruki klik na dugme *Hello world*

- Kreira novi događaj koji je vezan za dugme HelloWorld
 - *btnHelloWorld_Click*

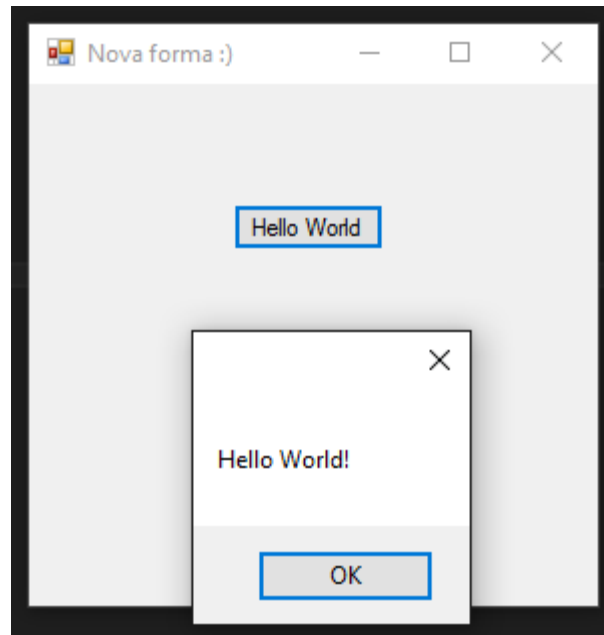
```
11 namespace WindowsFormsApplication1
12 {
13     2 references
14     public partial class FormNovaForma : Form
15     {
16         0 references
17         public FormNovaForma()
18         {
19             InitializeComponent();
20         }
21         1 reference
22         private void btnHelloWorld_Click(object sender, EventArgs e)
23         {
24         }
25     }
26 }
```

```
1 reference
private void btnHelloWorld_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello World!");
}
```



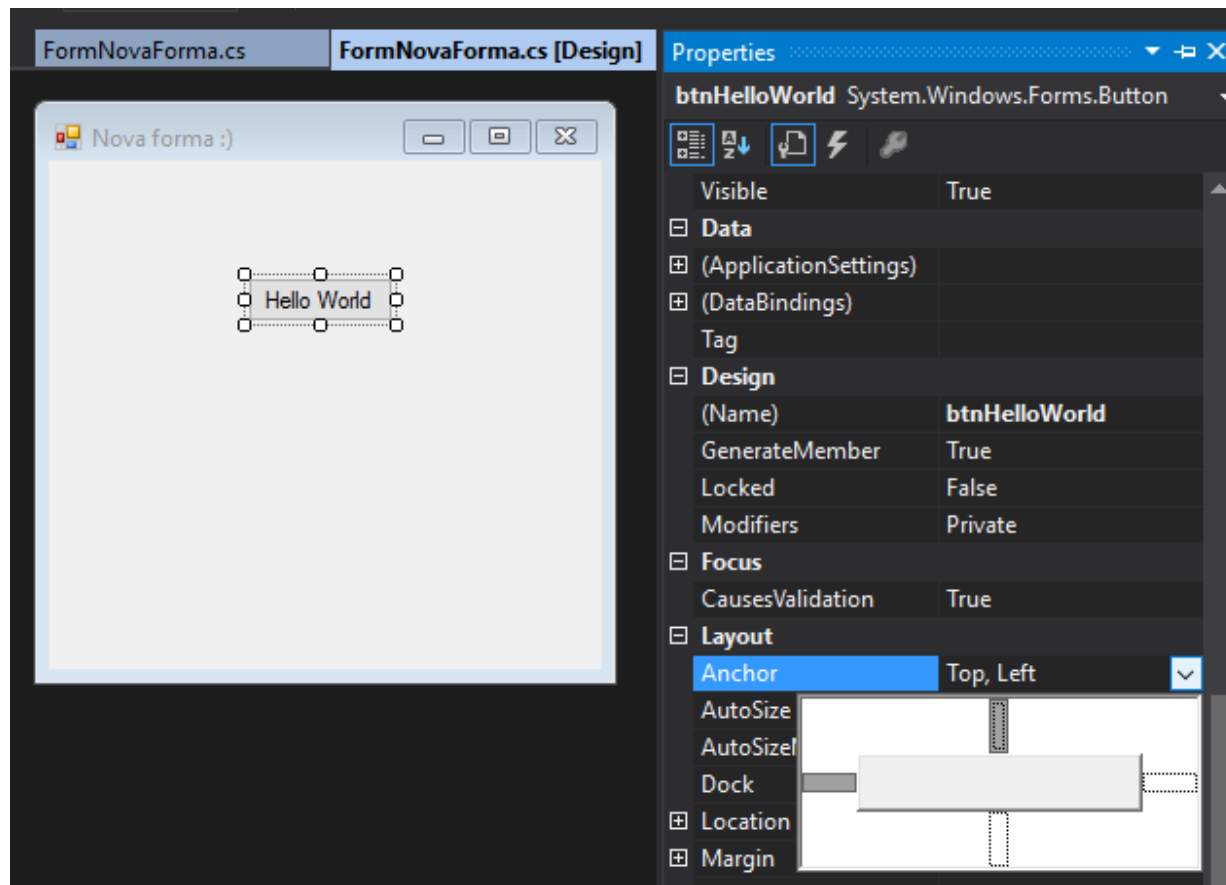
```
namespace WindowsFormsApplication1
{
    0 references
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        0 references
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FormNovaForma());
        }
    }
}
```


DODAVANJE DUGMETA NA FORMU



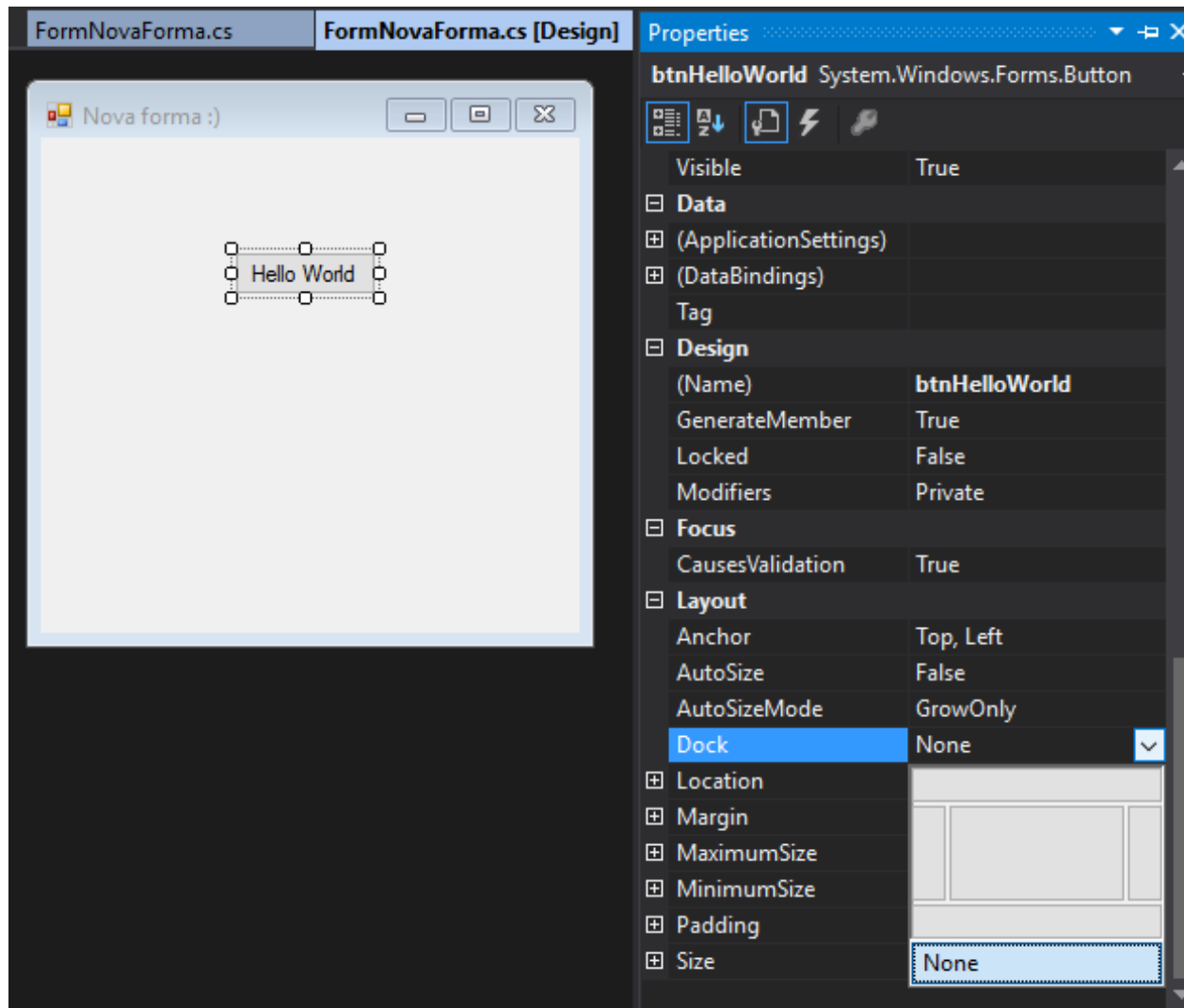
USIDRAVANJE *ANCHOR*

- Usidravanje komponente
 - Osigurava da će ivice kontrole ostati na istim mestima uzevši u obzir granice roditeljske kontrole/forme
- Kako bi se usidrila kontrola na formi
 - Postavi se njeno **Anchor** svojstvo
 - Default vrednosti: **Top, Left**
 - Ostali stilovi: **Bottom, Right**



FIKSIRANJE *DOCKING*

- Omogućuje da se ivice kontrole zalepe za ivice roditeljske kontrole.
- Kontrola se fiksira postavljanjem svojstva ***Dock*** u panelu ***Properties Window***.

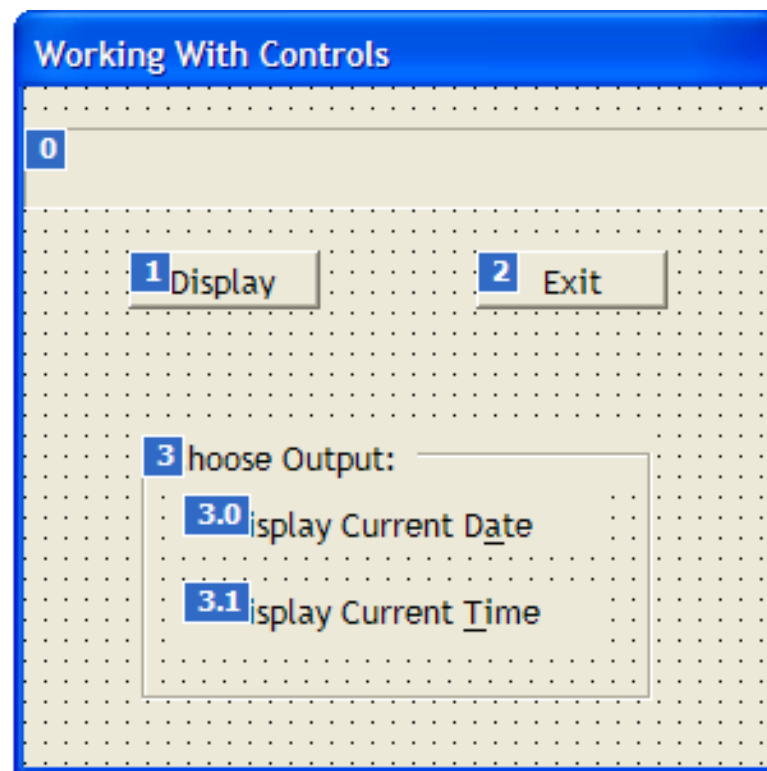


PODEŠAVANJE TAB ORDER-A ZA KONTROLE

- 1. način
 - U meniju **View**, odabrati **Tab Order**
 - Klik na kontrolu da bi se promenio njen tab order

-- || --

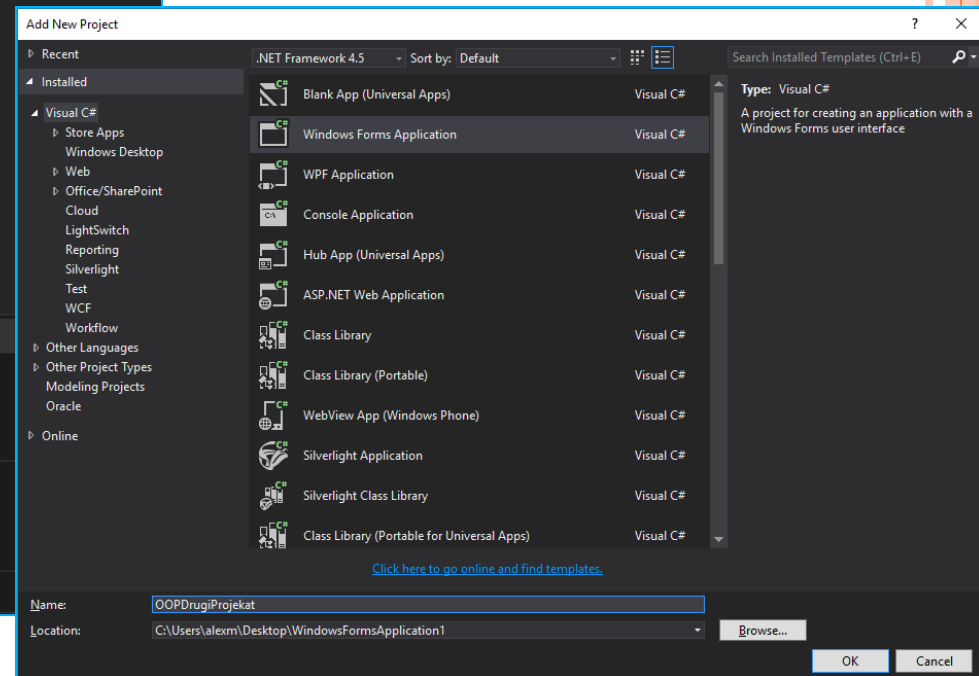
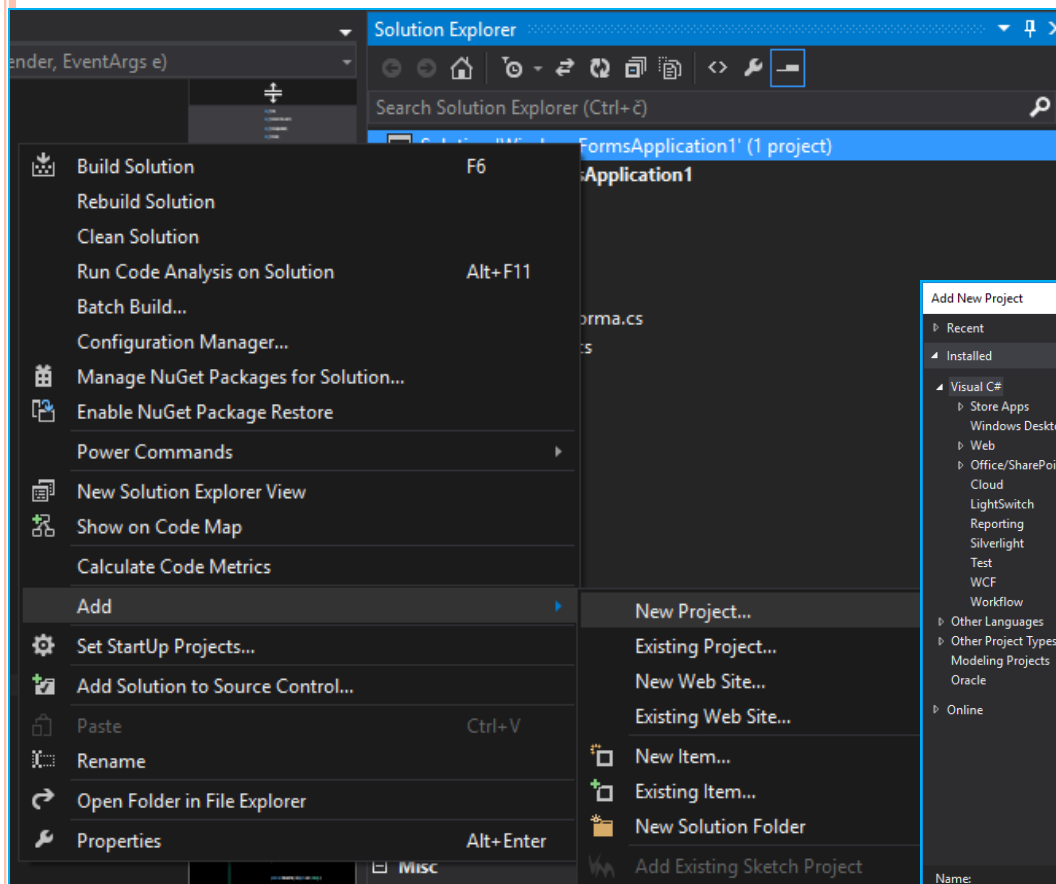
- Postaviti **TabIndex**
- Postaviti **TabStop** na vrednost **True**



VIŠE PROJEKTA U JEDAN SOLUTION

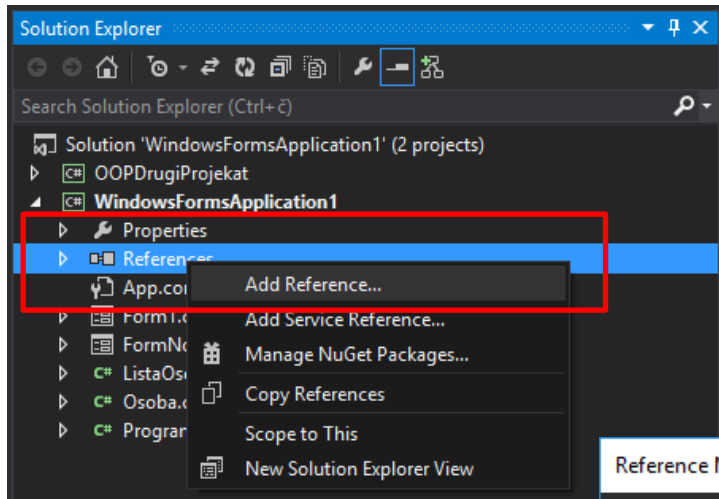
Dodavanje projekta u postojećem *Solution*-u

- Desni klik na *Solution* u panelu *Solution Explorer*
- Stavka **Add/New Project...**

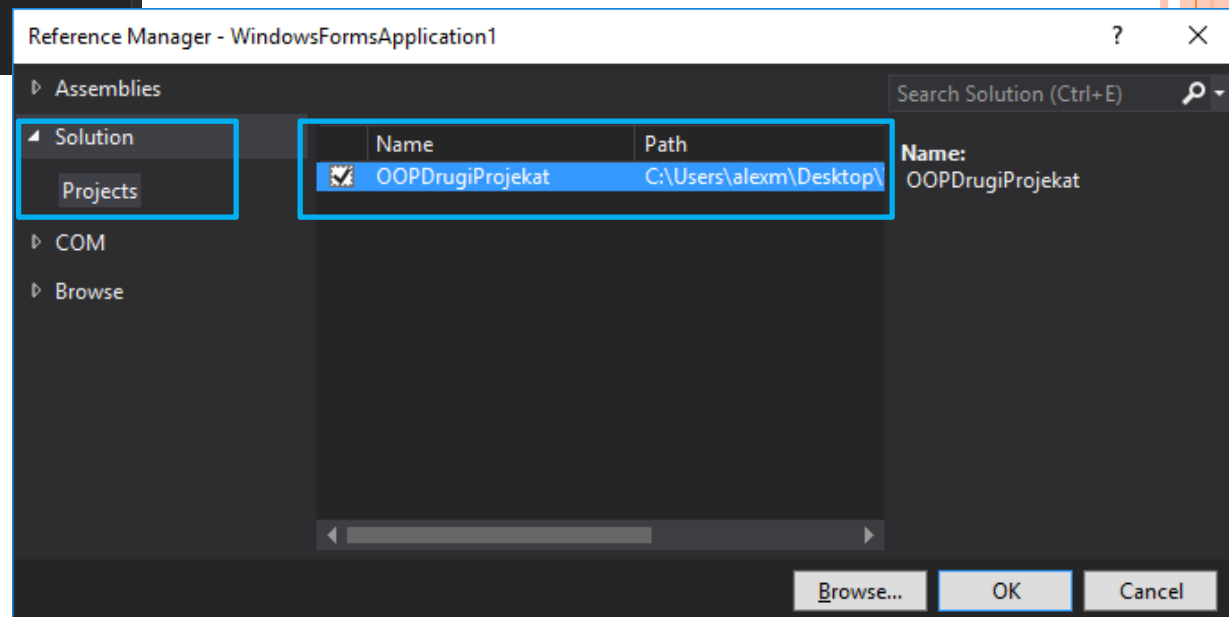


VIŠE PROJEKTA U JEDAN SOLUTION

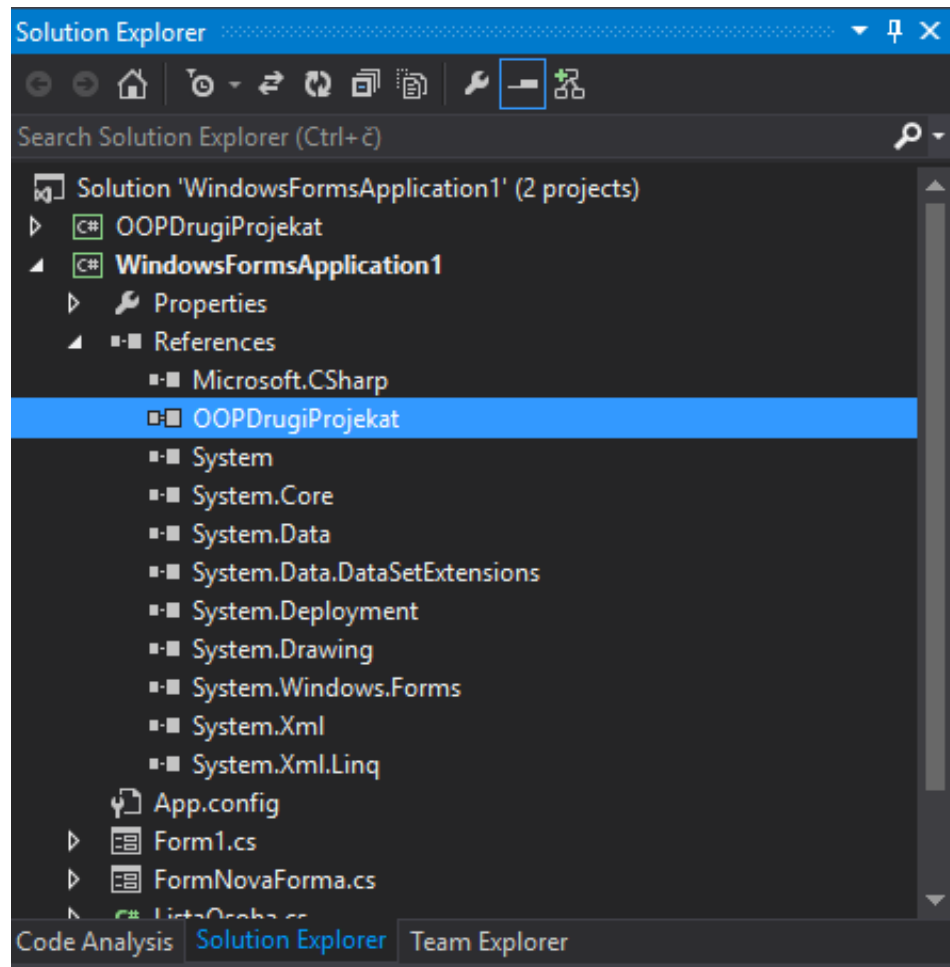
- Dodavanje reference na odabrani projekat iz istog *Solution-a*



- Odabere se projekat u kome se dodaje referenca na drugi projekat iz istog *Solution-a*
- Stavka *References*
- Desni klik na *References*
- Odabir stavke *Add Reference...*
- Izbor odgovarajućeg projekta u *Solution-u*.



VIŠE PROJEKTA U JEDAN SOLUTION



- Dodata referenca na projekat *OOPDrugiProjekat* u projektu *WindowsFormsApplication1* u okviru istog *Solution*-a *WindowsFormsApplication1*

PRIMER