# Akademija tehničko-vaspitačkih strukovnih studija



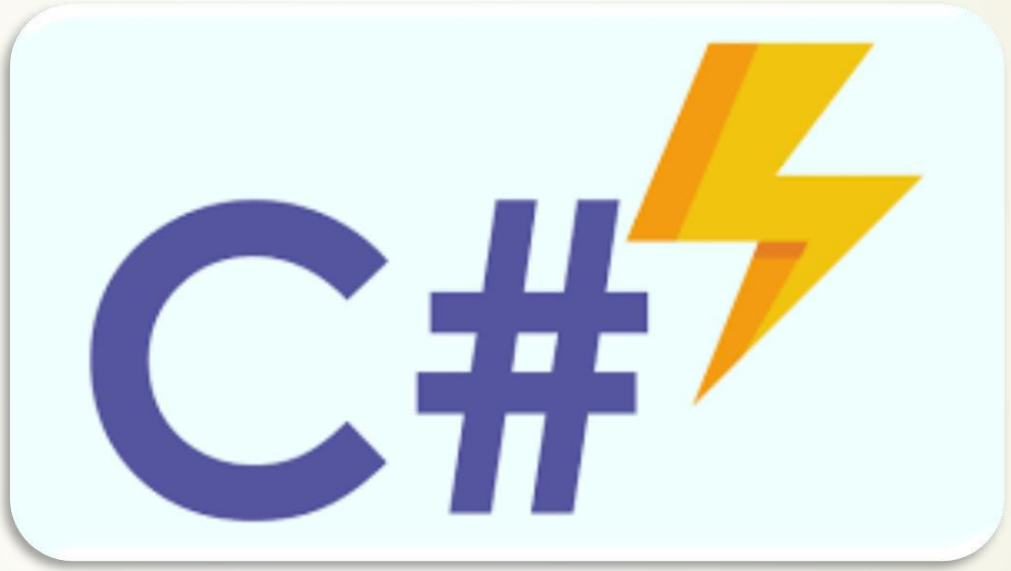

# .NET tehnologije

Prof. dr Zoran Veličković, dipl. inž. el.

2022/23.

Prof. dr Zoran Veličković, dipl. inž. el.

# .NET tehnologije




## Delegati i događaji u C#-u

(11)

# Sadržaj

# Događaji i akcije

- U **PROGRAMIRANJU** se često javljaju situacije kada treba izvršiti **NEKU AKCIJU**, ali unapred **NISU POZNATI** detalji oko toga **NA KOJI NAČIN** to uraditi!
- Zapravo, u tom trenutku **NE MORAJU BITI POZNATE** ni **METODE** ni **OBJEKTI** koje treba koristiti za izvođenje željene akcije.
- **POSLE PRITISKA** dugmeta, treba da se **KONTAKTIRA** neki objekt, ali unapred ne znamo **O KOM OBJEKTU** se tačno radi!
- **UMESTO** da se dugmetu odmah pridruži neki **KONKRETNI OBJEKT**, ono se povezuje sa **DELEGATOM** (specijalnom metodom) koji će ga tek u **TRENUTKU IZVRŠAVANJA** programa usmeriti na određenu - **KONKRETNU METODU**.
- Za realizaciju ove ideje u savremenim **OO** programskim jezicima razvijen je **MEHANIZAM** za stvaranje "**LABAVE VEZE**" između "**DOGAĐAJA**" i "**AKCIJE**" koju treba izvesti na pojavu definisanog događaja.
- Mehanizam povezivanje **DOGAĐAJA** i **METODA** je već pokazano u **`JavaScript`**-u, a na sličan način se primenjuje i u C#.

# Povezivanje događaja i akcija

- Prikazani koncept je izuzetno efikasan u **GRAFIČKOM RADNOM OKRUŽENJU**, gde svaka akcija korisnika (pritisak dugmadi, odabir opcije iz menija, pritisak ikonice, ...) izaziva neki **DOGAĐAJ** (engl. *event*).
- U grafičkom radnom okruženju kao što je npr. **Windows**, ovaj način programiranja se naziva **PROGRAMIRANJE ZASNOVANO NA DOGAĐAJIMA** (engl. *event-driven programming*).
- Pored **KORISNIČKI** izazvanih **DOGAĐAJA**, pomenutih prethodno, postoje i **SISTEMSKI DOGAĐAJI** koji se mogu aktivirati i bez direktne korisničke akcije (evo nekoliko primera: tik sistemskog časovnika, prijem e-pošte, ...).
- Informacije o **AKTIVIRANOM DOGAĐAJU**, **CLR** prvo **KAPSULIRA** u **SPECIJALIZOVANOM OBJEKTU DOGAĐAJA**, a potom se **POZIVA** **ODGOVARAJUĆA** **METODA** za njegovu obradu.
- U C#-u se ovo **POVEZIVANJE DOGAĐAJA** sa **METODAMA** za njegovu **OBRADU** (engl. *event handler*) realizuje pomoću **DELEGATA**.
- U C# **DELEGATI** su **OBJEKTI** osnovnih klasa **RADNOG OKRUŽENJA**!

# Delegati

- **DELEGATI** su **REFERENCNI TIPOVI** podataka koji **KAPSULIRAJU METODU** sa određenim **POTPISOM** i **POVRATNIM TIPOM** rezultata.
- Ako metoda za obradu događaja **PRIPADA NEKOJ INSTANCI**, onda, **DELEGAT** kapsulira i taj **ODREDIŠNI OBJEKAT**.
- **DELEGATI** omogućavaju **KAPSULIRANJE BILO KOJE METODE** koje poseduju **ISTI** "**POTPIS**".
- **DELEGATI** se kreiraju pomoću rezervisane reči "**`delegate`**" posle čega se navodi **TIP** povratnog **REZULTATA** i **POTPIS METODE** koje se mogu dodeliti delegatu:

```
delegate int myDelegate(int a, int b);
```

Ključna reč | Povratna vrednost | Ime delegata | Potpis metode

- U prikazanom primeru se deklariše **DELEGAT** sa imenom **`myDelegate`** koji može kapsulirati **BILO KOJU METODU** sa dve **`int`** promenljive kao argument i rezultatom tipa **`int`**.

# Kapsuliranje metode u delegatu

- Po deklerisanju delegata, **KAPSULIRA SE METODA ČLANICA** tako što se napravi **INSTANCA DELEGATA** sa **IMENOM METODE** kao **PARAMETROM**. Primer:

Kapsuliranje metode **add** u objektu delegata

```
myDelegate saberi = new myDelegate(add);
```

- Na ovaj način se obezbeđuje da se putem **DELEGATA** mogu pozivati **KAPSULIRANE METODE**.
- U nastavku će se ovaj delegat koristiti za kapsuliranje više (dve) metoda.

# Bazni primer delegata

```
delegate int myDelegate(int a, int b);

 int add(int a, int b) {
        return(a+b);
   }
 int sub(int a, int b) {
        return(a-b);
 }
myDelegate saberi = new myDelegate(add);
myDelegate oduzmi = new myDelegate(sub);
int a = saberi(4,3);          // a=7
int o = oduzmi (4,2);         // o=2
```

Dekleracija delegata **myDelegate**

Metode **add()** i **sub()** odgovaraju potpisu delegata **myDelegate**, te se **OBE** mogu kapsulirati prikazanim delegatom i po potrebi pozvati u trenutku izvršenja

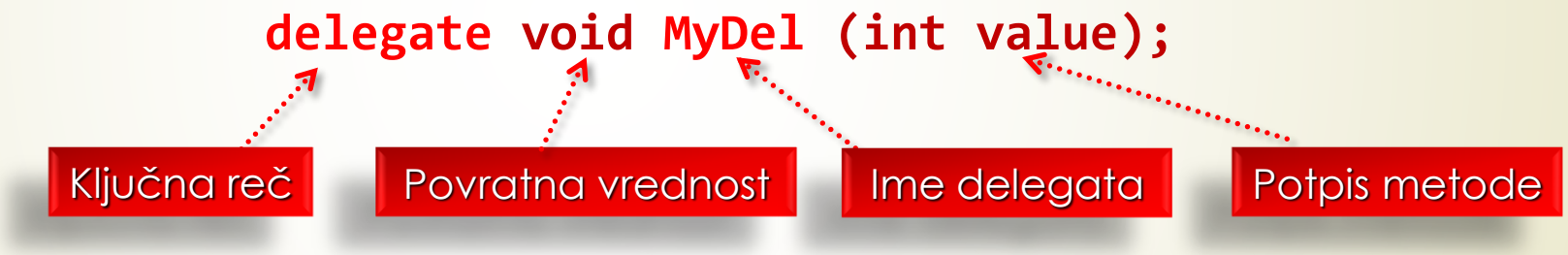
Za korišćenje delegata potrebno je napraviti **njegov OBJEKT**

Dva **objekta x** i **y** tipa **myDelegate** kapsuliraju odgovarjuće metode: **add** i **sub**

Poziv funkcija **add()** i **sub()** putem objekta **x** i **y**

# Razdvajanje delegata i događaja

- Pomoću delegata se **RAZDVAJAJU KLASE** koje ga **KORISTE** od klase koja ga **DEKLARIŠU**.
- Primer: Deklaracija delegata sa imenom **MyDel**:




- U nastavku će se ovaj delegat koristiti za kapsuliranje dve metode: **PrintLow()** i **PrintHigh()**.

# Primer delegata (1)

```
class Program
 {
   void PrintLow(int value)
       {
          Console.WriteLine( "{0} - Low Value", value );
       }
   void PrintHigh(int value)
      {
         Console.WriteLine( "{0} - High Value", value );
      }
```

Dekleracija klase **Program** koja svojim metodama **PrintLow** i **PrintHigh** može da podrži delegat **MyDel**

Dve metode klase **Program**: **PrintLow()** i **PrintHigh()**

# Primer delegata (2)

```
public void static Main( )
   {
     Program program = new Program();
     MyDel deleg_at;
    Random rand = new Random();
    int randomValue = rand.Next(99);
    deleg_at = randomValue < 50
        ? new MyDel (Program.PrintLow)
        : new MyDel (Program.PrintHigh);
    deleg_at(randomValue);
    }
}    // end Program
```

Kreirane objekta **Program**

Kreiranje delegat promenljive **deleg_at**

Kreiranje slučajnog broja između **0** i **99**

Kreiranje delegat objekta koji sadrži **PrintLow** ili **PrintHigh** i označiti ga kao **deleg_at**

Poziv **ODGOVARAJUĆE** metode delegatom

Pre izvršenja programa **NE ZNA** se koja će od metoda biti pozvana!

# Delegat kao parametar metode

```
public delegate void Print(int value);
    static void Main(string[] args)
    {
        PrintHelper(PrintNumber, 10000);
        PrintHelper(PrintMoney, 10000);
        Console.ReadLine();
    }
    public static void PrintHelper(Print delegateFunc, int numToPrint)
     {
        delegateFunc(numToPrint);
     }
    public static void PrintNumber(int num)
     {
        Console.WriteLine("Number: {0,-12:N0}", num);
     }
    public static void PrintMoney(int money)
     {
        Console.WriteLine("Money: {0:C}", money);
     }
```

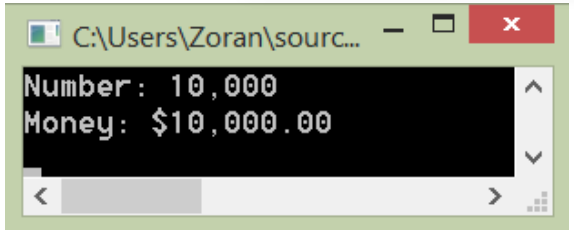
Deklaracija delegata **Print**

Metoda **PrintHelper** prosleđuje odgovarajuću **METODU** i **PARAMETAR**

Delegat **Print** kao parametar metode

Metoda **PrintHelper** prihvata delegat i integer kao parametre

Metode **PrintNumber** i **PrintMoney** koje zamenjuje delegata



# Generički delegat

```
public delegate T Transformacija<T>(T arg);
    public class Util
    {
        public static void Transform<T>(T[] values, Transformacija<T> t)
        {
            for (int i = 0; i < values.Length; i++)
                values[i] = t(values[i]);
        }

    }
    static void Main(string[] args)
    {
        int[] values = { 1, 2, 3, 4, 5 };
        Util.Transform(values, Square);
        foreach (int i in values)
            Console.Write(i + " ");
        Console.ReadLine();
    }
    static int Square(int x) => x * x;
```

Delegat **Transformacija**

Niz i generički delegat (metoda) kao argumenti

Metoda **Transform** radi sa prosleđenim tipom

Skraćena notacija - **LAMBDA IZRAZI**, kasnije nešto više

Metoda **Square()** vraća kvadrat parametra

# Delegat sortiranja

- Pozivom **DELEGATA** kome se **PROSLEĐUJU PARAMETRI** (u narednom primeru - dva objekta člana) se prepušta odgovornost **SORTIRANJA** metodi koja je **KAPSULIRANA DELEGATOM** (ovo će omogućiti da se različiti objekti mogu i različito sortirati):

```
public delegate Comparison WhichIsFirst (T obj1, T obj2)
```


- U ovoj definiciji **DELEGAT** `WhicIsFirst` **KAPSULIRA SVAKU METODU** čiji su parametri **DVA OBJEKTA**, dok je **REZULTAT** tipa `Comparasion`.
- Na ovaj način se može implementirati **VIŠE** različitih metoda za poređenje **VIŠE** različitih objekata.

# Višeznačni (višesmerni) delegati

- Jasno je da se **JEDNIM DELEGATOM** mogu kapsulirati **DVE ILI VIŠE** metoda.
- Ovo je izuzetno važno kod **OBRADE DOGAĐAJA**, npr. na pritisak dugmeta mogu se izvršiti **VIŠE** različitih akcija (metoda).
- Dakle, cilj je da se napravi **JEDAN DELEGAT** koji može pozivati **VIŠE KAPSULIRANIH METODA**.
- **DVA DELEGATA** se mogu **KOMBINOVATI** pomoću **OPERATORA SABIRANJA** ("**+**") čime se formira **VIŠEZNAČNI** (višesmerni) (engl. *multicast*) **DELEGAT**.
- **VIŠEZNAČNI DELEGAT** objedinjuje **SVE IZVORNE** delegatske metode.
- Ako su **`Writer`** i **`Logger`** delegati, oni se mogu **KOMBINOVATI** na sledeći način:

```
myMulticastDelegate = Writer + Logger;
```

- Primer korišćenja **OBJEDINJENOG OPERATORA** sabiranja i jednakosti ("**+=**") za dodeljivanje delegata **`Transmitter`** višesmernom delegatu **`myMulticastDelegate`**:

```
myMulticastDelegate += Transmitter;
```

# Događaji

- **DOGAĐAJI** (engl. *events*) u C# predstavljaju **MEHANIZAM** koji omogućava **KLASI** ili **OBJEKTU** da **OBAVESTE DRUGE KLASE** ili **OBJEKTE** da se **DESILO NEŠTO** od njihovog interesa (prethodno treba obaviti registracija na željeni događaj).
- **KLASE** koje **ŠALJU** (ili pokreću) mehanizam događaja nazivaju se **OBJAVLJIVAČI**, a klase koje **PRIMAJU** (ili rukuju) događajem nazivaju se **PRETPLATNICI**.
- U tipičnoj `WindowsForm` ili `WindowsWeb` C# aplikaciji, **PRETPLATNICI** se povezuju na **DOGAĐAJE** koje pokreću **Windows kontrole** (primer: dugmadi, kutije sa listama, labele, …).
- **PREGLED** i **POVEZIVANJE** već **OBJAVLJENIH DOGAĐAJA** od starane Windows kontrola se jednostavno obavlja u **IDE** Visual Studiu.
- IDE Visual Studio-a **AUTOMATSKI** dodaje:
  - **PRAZAN METOD OBRADE DOGAĐAJA** i
  - **PROGRAMSKI KOD** za **PRETPLATU** na događaj.

# Ugrađeni događaji

- **DOGAĐAJI** imaju sledeće karakteristike:
  - Klasa **OBJAVLJIVAČ** određuje kada je događaj aktiviran (kaže se još podignut), a **PRETPLATNICI** sami određuju **ŠTA** se preduzima kao odgovor na događaj.
  - Događaj može imati **VIŠE PRETPLATNIKA**, takođe, pretplatnik može da obradi **VIŠE DOGAĐAJA** od **VIŠE** objavljivača.
  - Događaji koji nemaju pretplatnike nikada **NEĆE** biti aktivirani!
  - U **Windowsu** se događaji obično koriste za **SIGNALIZACIJU** korisničkih AKCIJA kao što su **KLIKOVI** ili **IZBOR MENIJA** na grafičkim korisničkim interfejsima.
  - Kada neki događaj ima **VIŠE PRETPLATNIKA**, **RUKOVAOCI DOGAĐAJA** se **SINHRONO** (istovremeno) **POZIVAJU** kada se događaj aktivira.
  - U biblioteci klasa .NET Framework **DOGAĐAJI** se zasnivaju na **PREDEFINISANOM DELEGATU** `EventHandler` i **BAZNOJ KLASI** `EventArgs`.
  - `EventHandler` je **UNAPRED DEFINISANI** delegat za događaje koji **NE GENERIŠU** podatke.

# Povezivanje događaja i delegata (1)

- Deklaracija delegata **EventHandler** u .NET-u je sledeća:

Objekt **e** tipa **EventArgs** – nosi podatke o događajima

```
public delegate void EventHandler(object sender, EventArgs e);
```

**Objavljivač** (sender) – izvor događaja

- **MODEL DOGAĐAJA** u .NET Framevork-u zasnovan je na **DELEGATU DOGAĐAJA** koji **POVEZUJE DOGAĐAJ** sa njegovim korisnikom - **PRETPLATNIKOM**.
- Za **POKRETANJE DOGAĐAJA** neophodni su sledeći elementi:
  - **DELEGAT** koji **IDENTIFIKUJE METODU** koja **ODGOVARA NA DOGAĐAJ**;
  - **KLASA** (zapravo objekt) koja sadrži **PODATKE O DOGAĐAJU** - ako događaj ne generiše podatke, drugi parametar je vrednost polja **EventArgs.Empti**.
- **DELEGAT** je tip koji definiše **POTPIS**, odnosno tip **POVRATNE VREDNOSTI** i liste tipova parametara za metod.
- Može se koristiti tip delegata da se deklariše promenljiva koja može da se odnosi na **BILO KOJI METOD** sa istim potpisom kao i delegat.

# Povezivanje događaja i delegata(2)

- Već je napomenuto da **DOGAĐAJ** u Windowsu može izazvati **BILO KOJI ELEMENT RADNOG OKRUŽENJA** (npr. pritisak dugmeta izaziva događaj **`Click`**).
- **DOGAĐAJI** se implementiraju pomoću **DELEGATA** u tri koraka:
    1. Kreira se **KLASA OBJAVLJIVAČ DOGAĐAJA** kojom se i **DEKLARIŠE DELEGAT**;
    2. Kreira se **PRETPLATNIČKA KLASA** koja mora da poseduje **METODU SA POTPISOM** delegata;
    3. Pravi se **INSTANCA DELEGATA** koja **KAPSULIRA TU METODU**.
- Ova procedura obezbeđuje da se na **POJAVU DOGAĐAJA** pozivaju **METODE PRETPLATNIČKIH KLASA** i to pomoću delegata.
- Metoda koja **OBRAĐUJE DOGAĐAJ** zove se **PROCEDURA ZA OBRADU DOGAĐAJA** (engl. *event hendler*).
- **METODE ZA OBRADU DOGAĐAJA** se **DEKLARIŠU KAO DELEGATI**!

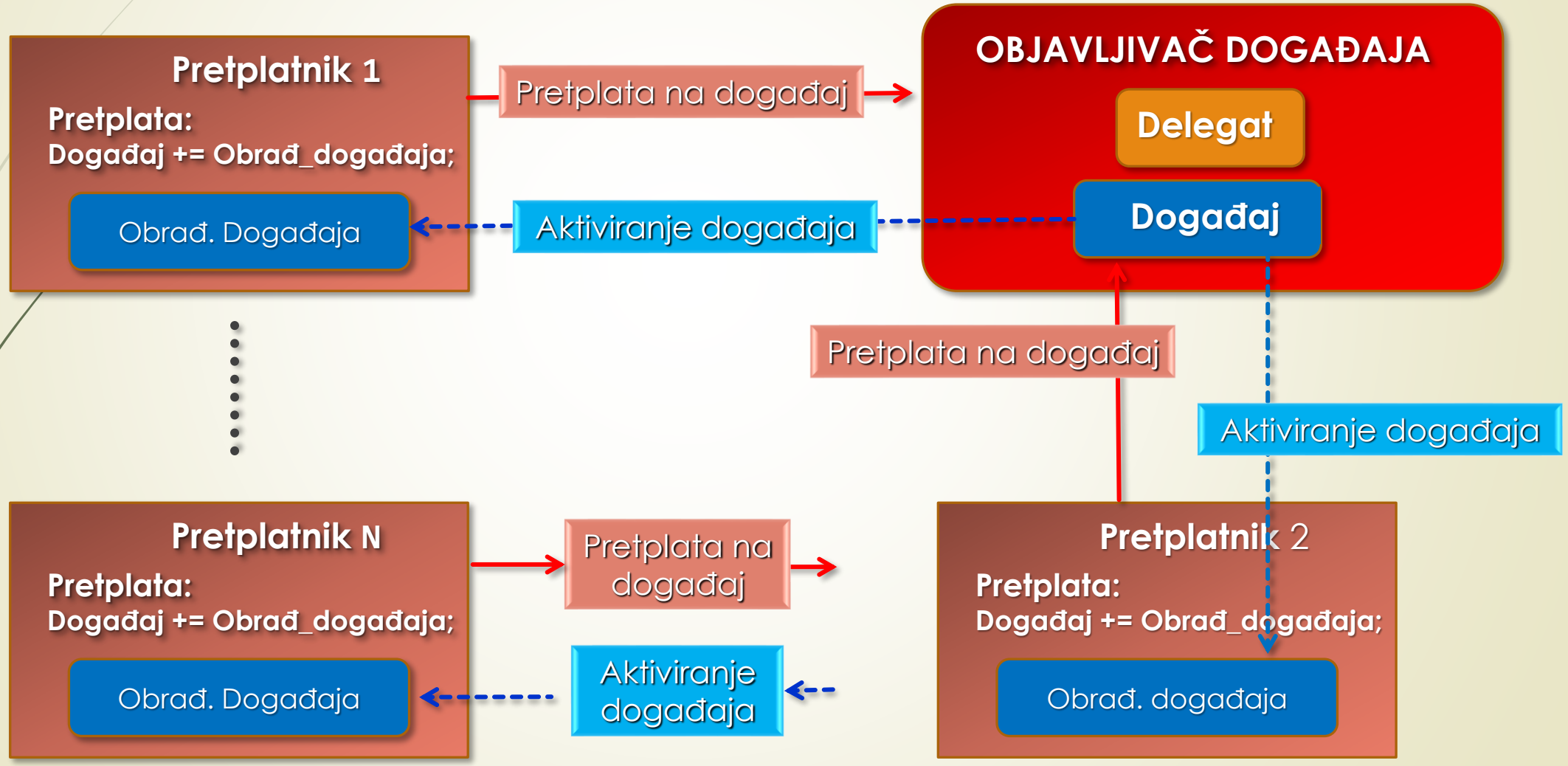
# Objavljivanje i pretplata (1)

- Klasa koja **IZAZIVA DOGAĐAJ** **NE ZNA** (ili ne mora znati) ništa o klasi koja će **OBRADITI** (reagovati) na događaj!
- Svaki OBJEKT može da "**OBJAVI**" **SKUP DOGAĐAJA** na koje se druge klase mogu "**PRETPLATITI**".
- Kada se desi objavljeni događaj, "**OBAVEŠTAVAJU**" se **SVE** **PRETPLAĆENE KLASE**.
- Tako, dugme - objavljivač (engl. *publisher*) bi moglo da obavesti **PROIZVOLJAN BROJ KLASA** pretplatnika (engl. *subscribers*) da je **AKTIVIRAN DOGAĐAJ** `Click`.
- Pretplatničke klase **NE MORAJU** da znaju kako radi klasa objavljivač!
- **OBJAVLJIVAČ** i **PRETPLATNIK** se razdvajaju pomoću **DELEGATA**.
- Ovo obezbeđuje da ove klase rade **NEZAVISNO JEDNE OD DRUGIH** što ima za posledicu pojednostavljenje **ODRŽAVANJE KODA**.

# Objavljivanje i pretplata (2)

- Već znamo, **PO KONVENCIJI** u .NET-u procedure za obradu događaja prihvataju **DVA PARAMETRA** i vraćaju rezultat tipa **`void`**.
- **PRVI PARAMETAR** u proceduri .NET-a za obradu događaja je "**IZVOR DOGAĐAJA**" (objekt koji izaziva događaj), dok je **DRUGI PARAMETAR** obavezno **OBJEKAT IZVEDEN IZ KLASE `EventArgs`**.
- Klasa **`EventArgs`** je osnovna **PREDEFINISANA KLASA** koja **KAPSULIRA** (sadrži) sve podatke o događaju.
- Klase se **PRETPLAĆUJU** na događaje putem **DELEGATA**, odnosno **PONIŠTAVAJU PRETPLATU** samo pomoću **OBJEDINJENIH OPERATORA**: ("**`+=`**") i ("**`-=`**").
- **DELEGATSKE METODE** se deklarišu rezervisanom rečju "**`event`**", i **NE MOGU** se direktno pozvati ali se na njih može "**PRETPLATITI**".
- U tom slučaju, **DELEGAT MOŽE BITI POZVAN** samo od klasa koje su ga definisale, a ostale klase se mogu **PETPLATITI** na njega.

# Objavljivanje i pretplata (3)



# Događaji Windows kontrola (1)

- **DELEGAT** se ponaša kao **POINTER NA FUNKCIJU**, a koristi se da se **POZOVE METODA** koja je specificirana u **TOKU IZVRŠENJA**.
- Na početku predavanja je konstatovano da se delegati koriste u slučajevima kada se **NE ZNA UNAPRED** koja metoda treba da se pozove.
- Već smo konstatovali, **DELEGATI** i **DOGAĐAJI** su tesno povezani.
- **DOGAĐAJI** se koriste da **OBAVESTE ZAINTERESOVANE METODE** da se **NEŠTO DESILO** u odgovarajućem objektu.
- Primer događaja u Wundows-u je **PRITISAK DUGMETA** ili **SELEKCIJA OPCIJE** iz menija.
- Pogledajmo **PRIMER DELEGATA** standardnih **WINDOWS KONTROLA** (dugmeta) na jednostavnoj formi.
- Prvo treba napraviti **METODU** koja obrađuje **ODGOVARAJUĆI DOGAĐAJ** (u našem slučaju na pritisak dugmeta).

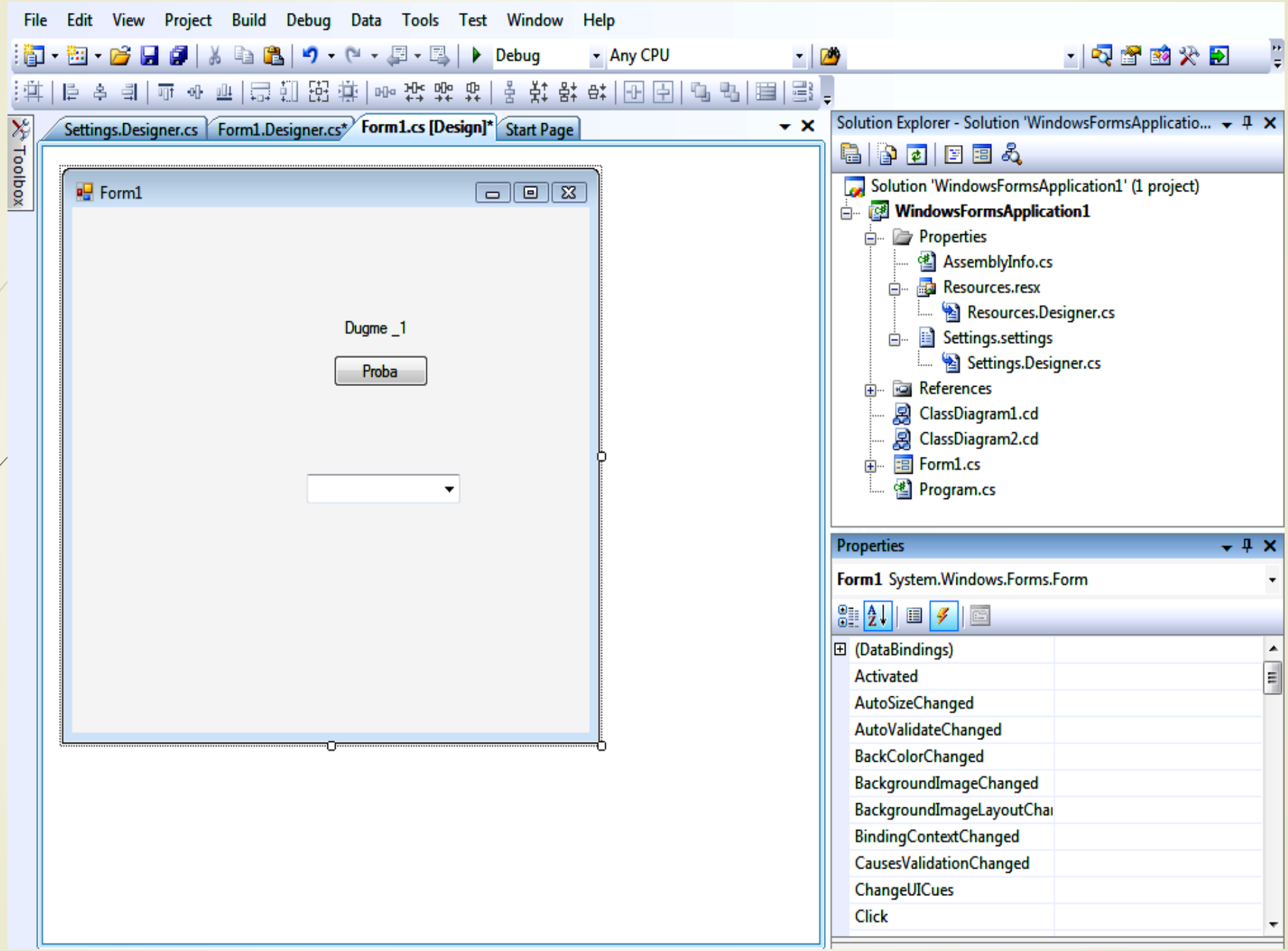
# Događaji Windows kontrola (2)

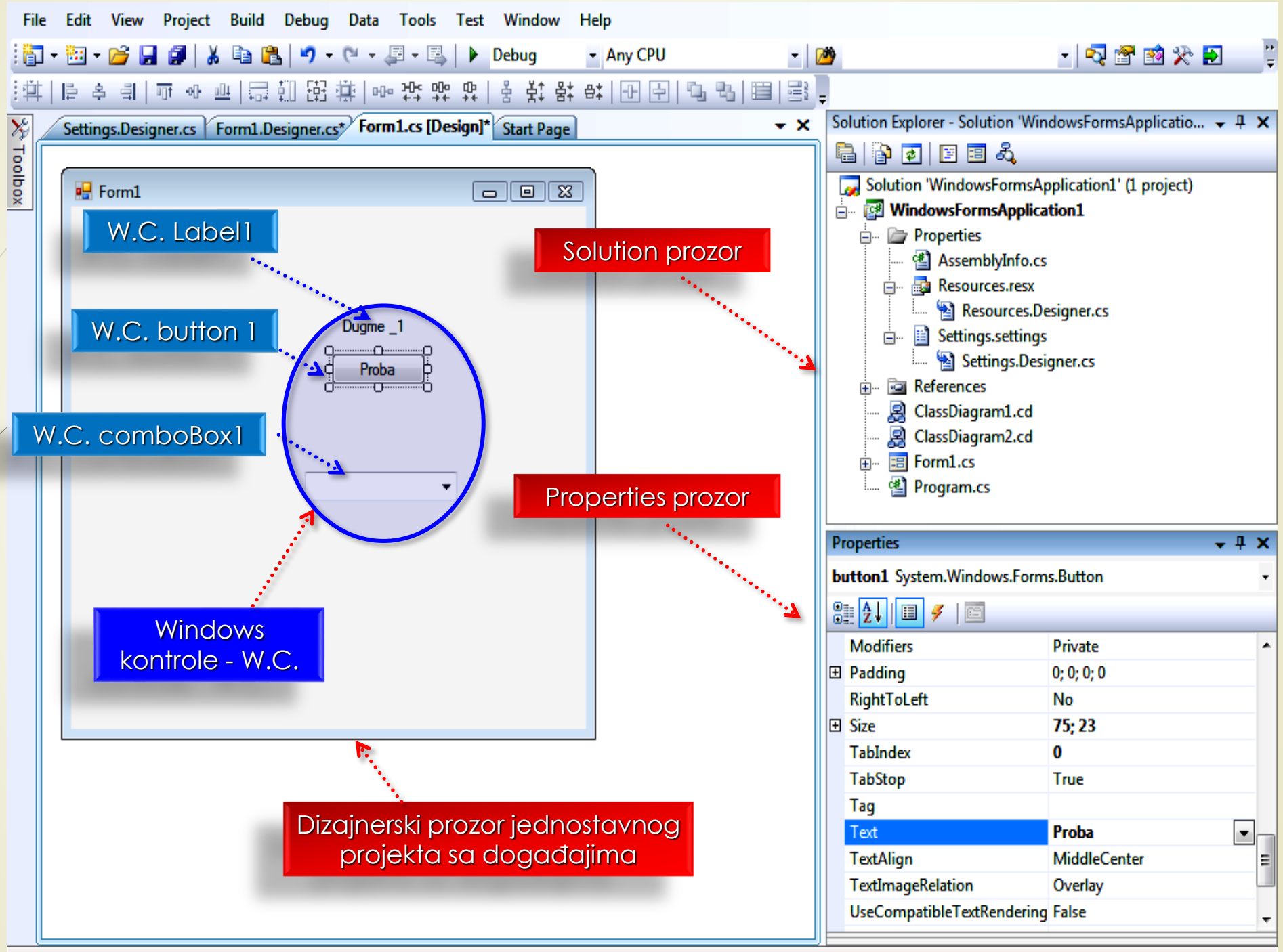
- **METOD ZA OBRADU** se povezuje sa **DOGAĐAJEM** tako što se napravi **INSTANCA** odgovarajućeg **DELEGATA** koji će rukovati događajem i iskoristi se **OPERATOR** (**+=**) za **POVEZIVANJE DOGAĐAJA I DELEGATA** koji **VEĆ POSTOJI**.
- **POVEZIVANJE VIŠESTRUKIH DOGAĐAJA** sa **JEDNIM RUKOVAOCEM** se obavlja operatorom (**+=**) na isti način kao pri dodavanju pojedinačnog rukovaoca.
- Za događaje vezane za Windows kontrole već su **UNAPRED DEFINISANI PODRAZUMEVANI DELEGATI**.
- Prvo treba napraviti **INSTANCU** podrazumevanog **DELEGATA** prilikom dodavanja novog rukovaoca ovim događajima.
- Klasa delegata **`System.EventHandler`** se koristi za većinu događaja iz prostora imena **`System.Windows.Forms`**:

Formiranje objekta

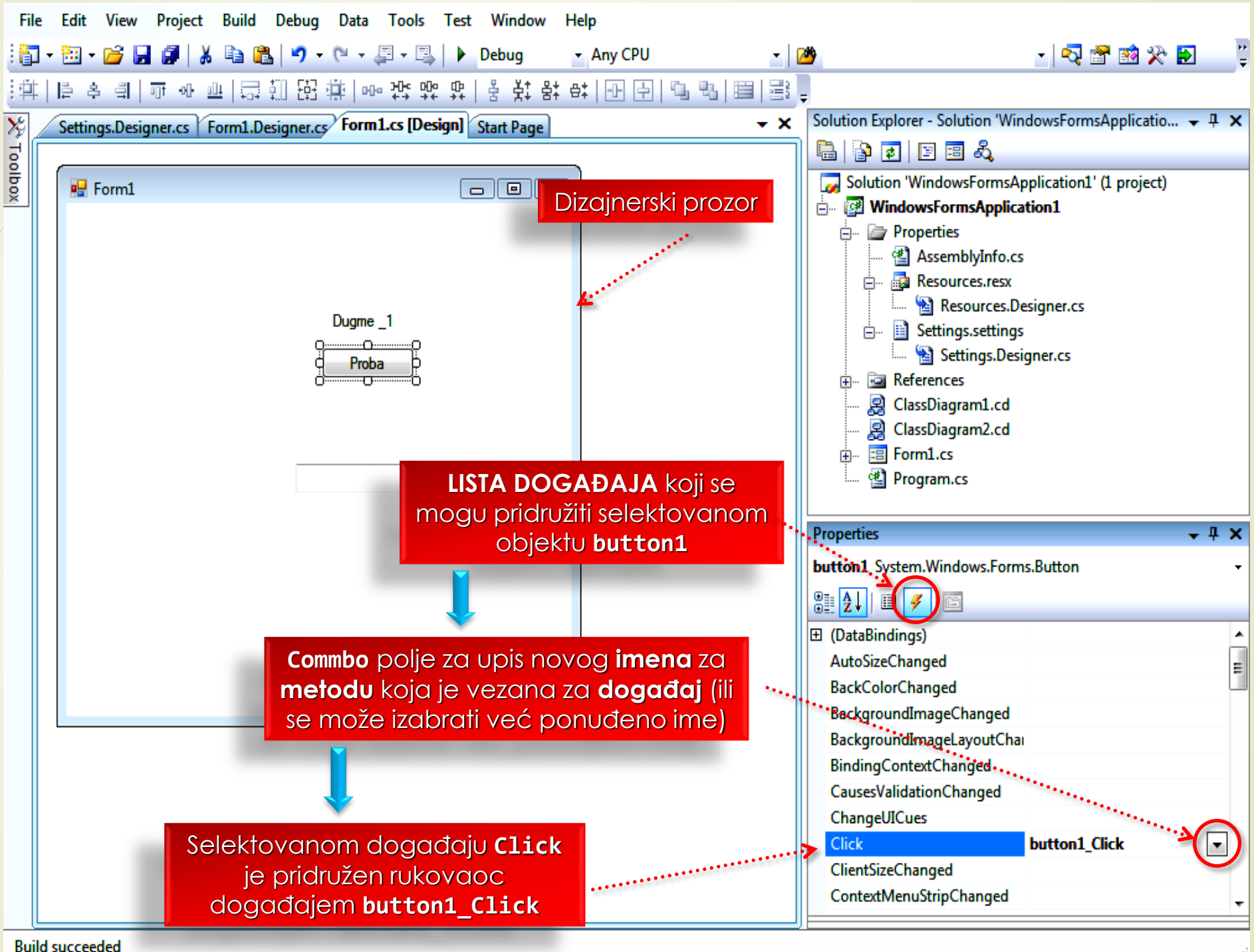
```
Button1.Click += new System.EventHandler(clickHandler)
```

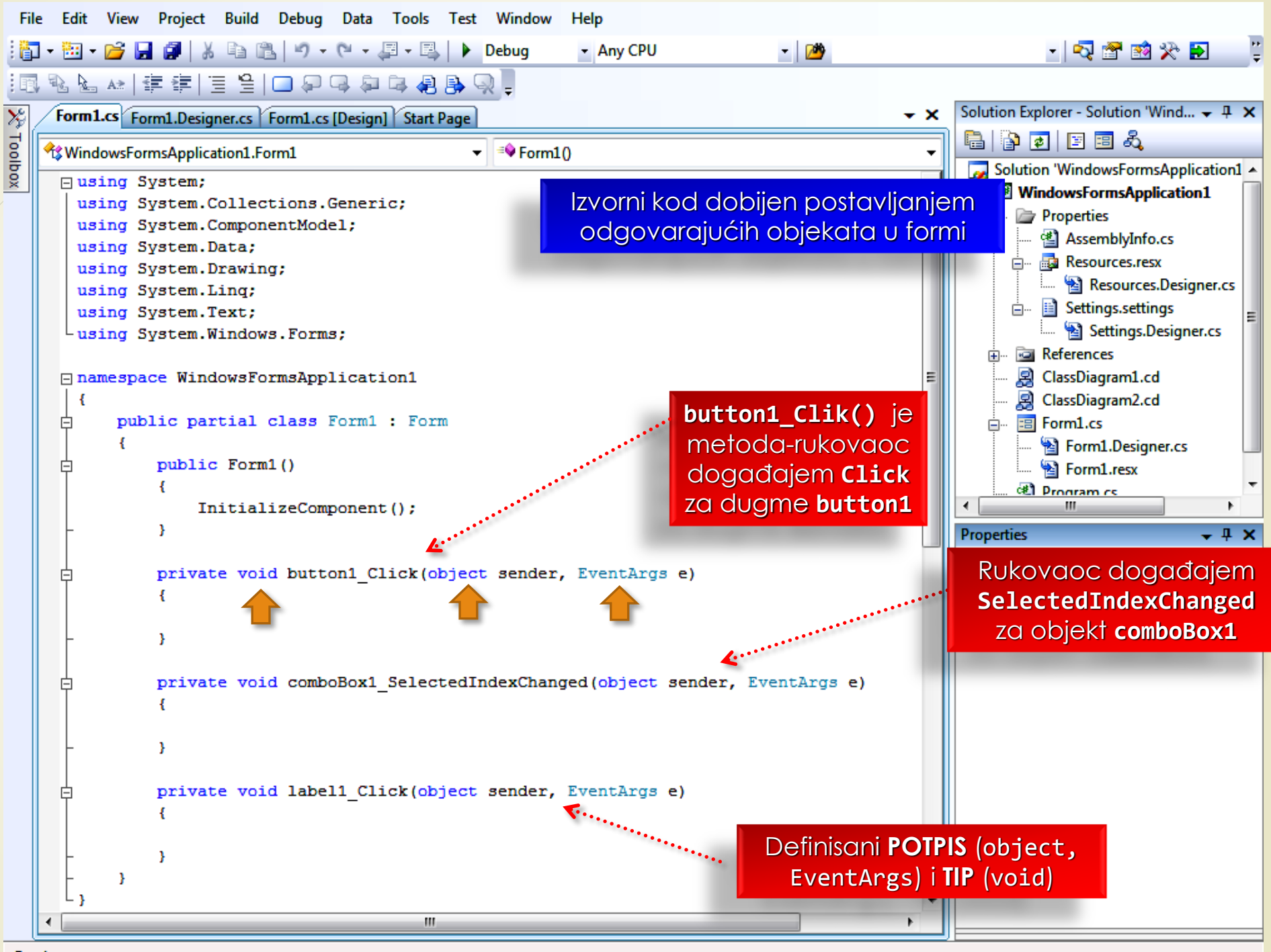
Kontrola — Događaj — Delegat — Metoda

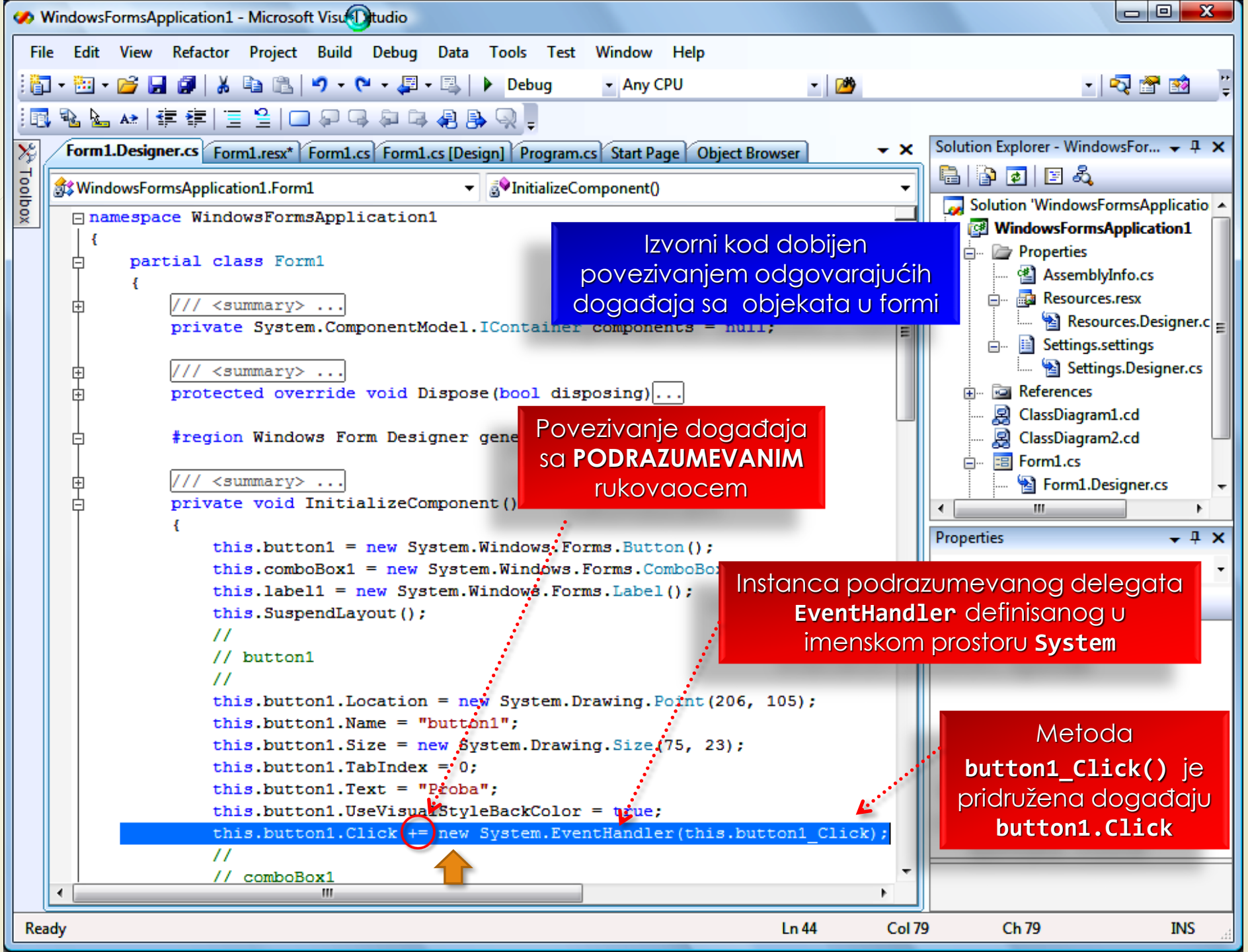

File Edit View Project Build Debug Data Tools Test Window Help
Debug
Any CPU
Toolbox
Settings.Designer.cs
Form1.Designer.cs*
Form1.cs [Design]*
Start Page
Form1
Dugme _1
Proba
Solution Explorer - Solution 'WindowsFormsApplicatio...
Solution 'WindowsFormsApplication1' (1 project)
WindowsFormsApplication1
Properties
AssemblyInfo.cs
Resources.resx
Resources.Designer.cs
Settings.settings
Settings.Designer.cs
References
ClassDiagram1.cd
ClassDiagram2.cd
Form1.cs
Program.cs
Properties
Form1 System.Windows.Forms.Form
(DataBindings)
Activated
AutoSizeChanged
AutoValidateChanged
BackColorChanged
BackgroundImageChanged
BackgroundImageLayoutChar
BindingContextChanged
CausesValidationChanged
ChangeUICues
Click




**W.C.** Windows Control


File Edit View Project Build Debug Data Tools Test Window Help
Debug
Any CPU
Settings.Designer.cs
Form1.Designer.cs
Form1.cs [Design]
Start Page
Toolbox
Form1
Dugme _1
Proba
Dizajnerski prozor
LISTA DOGAĐAJA koji se mogu pridružiti selektovanom objektu button1
Commbo polje za upis novog imena za metodu koja je vezana za događaj (ili se može izabrati već ponuđeno ime)
Selektovanom događaju Click je pridružen rukovaoc događajem button1_Click
Solution Explorer - Solution 'WindowsFormsApplicatio...
Solution 'WindowsFormsApplication1' (1 project)
WindowsFormsApplication1
Properties
AssemblyInfo.cs
Resources.resx
Resources.Designer.cs
Settings.settings
Settings.Designer.cs
References
ClassDiagram1.cd
ClassDiagram2.cd
Form1.cs
Program.cs
Properties
button1 System.Windows.Forms.Button
(DataBindings)
AutoSizeChanged
BackColorChanged
BackgroundImageChanged
BindingContextChanged
CausesValidationChanged
ChangeUICues
Click
button1_Click
ClientSizeChanged
ContextMenuStripChanged
Build succeeded


File Edit View Project Build Debug Data Tools Test Window Help
Debug
Any CPU
Form1.cs
Form1.Designer.cs
Form1.cs [Design]
Start Page
Toolbox
WindowsFormsApplication1.Form1
Form1()
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
}
private void button1_Click(object sender, EventArgs e)
{
}
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
}
private void label1_Click(object sender, EventArgs e)
{
}
}
}
Izvorni kod dobijen postavljanjem odgovarajućih objekata u formi
button1_Clik() je metoda-rukovaoc događajem Click za dugme button1
Rukovaoc događajem SelectedIndexChanged za objekt comboBox1
Definisani POTPIS (object, EventArgs) i TIP (void)
Solution Explorer - Solution 'Wind...
Solution 'WindowsFormsApplication1
WindowsFormsApplication1
Properties
AssemblyInfo.cs
Resources.resx
Resources.Designer.cs
Settings.settings
Settings.Designer.cs
References
ClassDiagram1.cd
ClassDiagram2.cd
Form1.cs
Form1.Designer.cs
Form1.resx
Program.cs
Properties


WindowsFormsApplication1 - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Data Tools Test Window Help
Debug
Any CPU
Form1.Designer.cs
Form1.resx*
Form1.cs
Form1.cs [Design]
Program.cs
Start Page
Object Browser
Toolbox
WindowsFormsApplication1.Form1
InitializeComponent()
namespace WindowsFormsApplication1
{
partial class Form1
{
/// <summary> ...
private System.ComponentModel.IContainer components = null;
/// <summary> ...
protected override void Dispose(bool disposing)...
#region Windows Form Designer gene
/// <summary> ...
private void InitializeComponent()
{
this.button1 = new System.Windows.Forms.Button();
this.comboBox1 = new System.Windows.Forms.ComboBo
this.label1 = new System.Windows.Forms.Label();
this.SuspendLayout();
//
// button1
//
this.button1.Location = new System.Drawing.Point(206, 105);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(75, 23);
this.button1.TabIndex = 0;
this.button1.Text = "Proba";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button1_Click);
//
// comboBox1
Izvorni kod dobijen povezivanjem odgovarajućih događaja sa objekata u formi
Povezivanje događaja sa PODRAZUMEVANIM rukovaocem
Instanca podrazumevanog delegata EventHandler definisanog u imenskom prostoru System
Metoda button1_Click() je pridružena događaju button1.Click
Solution Explorer - WindowsFor...
Solution 'WindowsFormsApplicatio
WindowsFormsApplication1
Properties
AssemblyInfo.cs
Resources.resx
Resources.Designer.c
Settings.settings
Settings.Designer.cs
References
ClassDiagram1.cd
ClassDiagram2.cd
Form1.cs
Form1.Designer.cs
Properties
Ready
Ln 44
Col 79
Ch 79
INS

# Interakcije sa mišem (1)

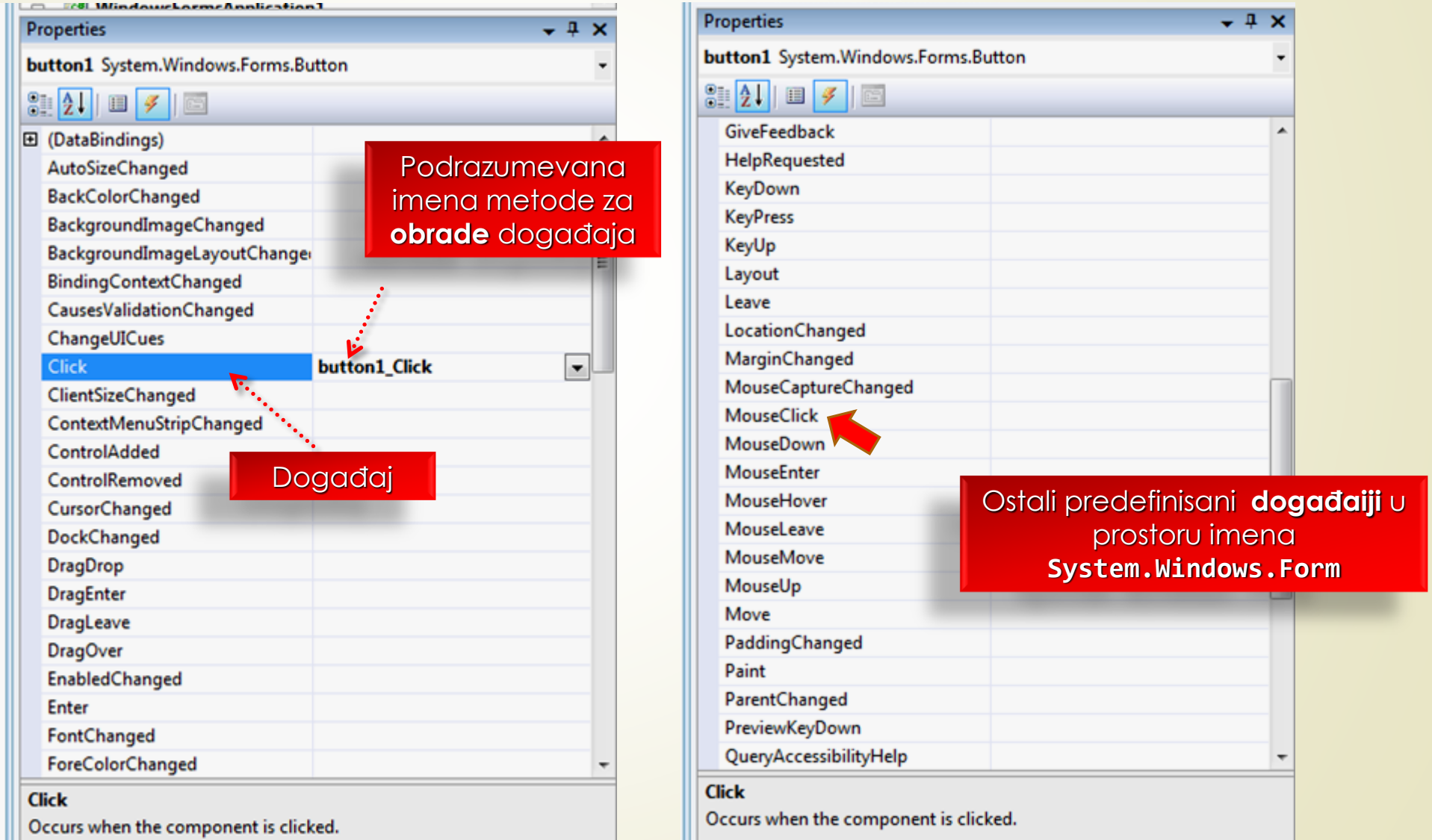
- **`WindowsForm` KONTROLE** su sposobne da **IZAZOVU DOGAĐAJ** koji označava **INTERAKCIJU SA MIŠEM** (reakcija na pritisak tastera mišem, pokazivač miša se nalazi iznad kontrole, ...).
- Tako, događaji **`Click`** i **`DoubleClick`** se uglavnom koriste za izvršavanje koda zasnovanog na **IZBORU KORISNIKA**.
- Kao i svi događaji, i oni **PROSLEĐUJU OBJEKT** klase **`EventArgs`** svojim rukovaocima kao i **REFERENCU NA OBJEKT** u kome se događaj desio.
- Tako, objekat **`MouseEventArgs`** sadrži informacije o:
  - Stanju MIŠA,
  - Lokacilji MIŠA.
- **SVOJSTVA** ovog objekta su data u sledećoj tabeli.

# Interakcije sa mišem: događaji i svojstva

| DOGAĐAJ | Funkcija |
|---|---|
| MouseEnter | Pokazivač miša ušao u kontrolu, EventArgs (EA). |
| MouseMove | Pokazivač miša prelazi iznad kontrole, MouseEventArgs |
| MouseHover | Pokazivač miša zastao na kontroli, EA. |
| MouseDown | Pokazivač miša iznad kontrole i dugme pritisnuto. |
| MouseWheel | Pomera se točak miša i kontola u fokusu, MouseEventArgs. |
| MouseUp | Pokazivač miša iznad kontrole i dugme otpušteno, MEA. |
| MouseLeave | Pokazivač miša se pomerio sa kontrole, EA. |

| SVOJSTVA | Opis |
|---|---|
| Button | Ovo svojstvo vraća koje je dugme pritisnuto. |
| Cliks | Ovo svojstvo vraća koliko je puta dugme miša pritisnuto. |
| Delta | Ovo svojstvo vraća broj "recki" za koje se rotirao točak miša. Broj može biti pozitivan ili negativan. |
| X | Ovo svojstvo vraća koordinatu **x** mesta na kome je miš pritisnut. |
| Y | Ovo svojstvo vraća koordinatu **y** mesta na kome je miš pritisnut. |

# Interakcije sa mišem (2)



# Anonimne f-je, delegati i lambda izrazi*

```
class Program
    {
        delegate double MyDel (int par);
        static void Main(string[] args)
        {
            MyDel del = delegate (int x) { return x + 1; };
            MyDel le1 = (int x) => { return x + 1; };
            MyDel le2 = (x) => { return x + 1; };
            MyDel le3 = x => { return x + 1; };
            MyDel le4 = x => x + 1;

             Console.WriteLine("{0}", del(12));
             Console.WriteLine("{0}", le1(12)); Console.WriteLine("{0}", le2(12));
             Console.WriteLine("{0}", le3(12)); Console.WriteLine("{0}", le4(12));
         }
    }
```

Primer **anonimne metode** u delegatu

Pretvaranje anonimne metode u lambda izraze (verzija 1)

lambda **operator (=>)**

(verzija 2)

(verzija 3)

(verzija 4)

Lambda izrazima se **pojednostavljuje** programski kod

# Korisnički događaji i delegati* (1)

- Već je napomenuto da po konvenciji u .NET-u, procedure za obradu događaja imaju **DVA PARAMETRA** i vraćaju rezultat tipa **`void`**.
- **PRVI PARAMETAR** u proceduri .NET-a za obradu događaja je "**izvor događaja**" (objekt koji izaziva događaj), dok je **DRUGI PARAMETAR** obavezno **objekat** izveden iz klase **`EventArgs`**.
- Klasa **`EventArgs`** je osnovna klasa koja **KAPSULIRA** (sadrži) sve podatke o događaju.
- Klasa **`EventArgs`** nasleđuje sve metode od klase **`Object`** i sadrži javno statičko polje **`EMPTY`**, a predstavlja **DOGAĐAJ BEZ STANJA**.
- Primer:
  - Biće napravljena **KLASA OBJAVLJIVAČ** "**`Clock`**" koja pomoću **delegata** obaveštava potencijalne pretplatnike o promeni lokalnog vremena za **jedan sekund**.
  - Ime ovog delegata je: **`SecondChangeHandler`**.
- * Nije obavezni deo osnovnog kursa

# Korisnički događaji i delegati* (2)

- Pogledajmo dekleraciju delegata **SecondChangeHandler**:

```
public delegate void SecondChangeHandler
                     (object Clock, TimeInfoEventArgs timeInformation);
```

- Ovaj delegat može kapsulirati svaku metodu koja vraća tip **void** i ima **DVA OBJEKTA** kao parametre.
- Prvi parametar je **objekt** tipa **Clock** (on izaziva događaj), a drugi **objekt** je tipa **TimeInfoEventArgs** i sadrži sve informacije korisne za klase koje su zainteresovane za ovaj događaj.
- Objekti klase **TimeInfoEventArgs** sadrže podatke o **TEKUĆEM VREMENU**: satu, minutu i sekundi kao i metodu **Run()**.
- Kod klase **TimeInfoEventArgs** koja **KAPSULIRA DOGAĐAJ** (MORA biti izvedena iz klase **EventArgs**) je dat na sledećem slajdu.

# Klasa TimeInfoEventArgs*

```
public class TimeInfoEventArgs : EventArgs
{
    public TimeInfoEventArgs (int hour, int minute, int second)
      {
        this.hour = hour;
        this.minute = minute;
        this.second = second;
      }
     public readonly int hour;
     public readonly int minute;
     public readonly int second;
}
```

Mora biti izvedena iz **EvenrArgs** (kapsulira događaj)

Konstruktor

Podaci o tekućem satu, minutu i sekundi.

Javne celobrojne promenljive samo za čitanje.

Objekti klase **TimeInfoEventArgs** sadrže podatke o **tekućem satu, minutu i sekundi**.

# Klasa Clock*

```
// Klasa Clock je OBJAVLJIVAČ DOGAĐAJA koji će pretplatnici osluškivati
// Klasa Clock objavljuje događaj: OnSecondChange
// Zainteresovani pretplatnici treba da se pretplate na ovaj događaj
public class Clock
{
   private int hour, ... minute, second;
      public delegate void SecondChangeHandler
                  (object Clock, TimeInfoEventArgs timeInformation);
. . .
      public event SecondChangeHandler OnSecondChange;
// Pokreće časovnik posle svake sekunde, izaziva se događaj
 // Klasi Clock je dodata funkcija Run() ...
```

Klasa objavljivač

DEKLERACIJA delegata koji **PRETPLATNICI moraju** implementirati.

Dva parametra delegata

Rezervisana reč

Formira se **instanca OnSecondChange** delegata **SecondChangeHandler**

# Metoda Run* (1)

```
public void Run()
{
    for(;;)
    {
       Thread.Sleep(10);
       System.DateTime dt = System.DateTime.Now;
         if(dt.Second != second)
          {
            TimeInfoEventArgs timeInformation =
                    new TimeInfoEventArgs(dt.Hour, dt.Minute,dt.Second);
```

Dekleracija niti, kasnije više o nitima.

Pravi se beskonačna petlja!

Miruje 10 millisekundi

čita tekuće vreme

Ako se vrednost sekundi promenila obavesti pretplatnike!

Kreiranje objekta **timeInformation** tipa **TimeInfoEventArgs** da bi se **prosledio** pretplatnicima.

# Metoda Run* (2)

```
// ako ima pretplatnika obaveštava ih

if (OnSecondChange != null)
    {
        OnSecondChange(this, timeInformation);
    }
  }

// ažurira stanje promenljivih za vreme
    this.second = dt.Second;
    this.minute = dt.Minute;
    this.hour = dt.Hour;
 }
}
```

Ako ima pretplatnika obavesti ih!

Izazivanje događaja **OnSecondChange**

**Prvi** parametar: Tekući objekt, izazivač događaja

**Drugi** parametar: Objekat kojim se obaveštavaju pretplatnici tako što se izaziva događaj **OnSecondChange**

# Pretplata na korisnički događaj* (1)

- Sada će biti napravljene **DVE KLASE** koje se mogu **PRETPLATITI** na ovaj događaj:
  - **`DisplayClock`** i
  - **`LogCurrentTime`**.
- Klasa **`DisplayClock`** ima samo dve metode:
  - **`Subscribe()`** metoda koja je pretplaćena na događaj **`OnSecondChange`**.
  - Funkcija za obradu događaja:

```
TimeHasChanged(object theClock, TimeInfoEventArgs ti)
```

- Kada se pozove metoda **`Subscribe()`**, ona pravi novi delegat tipa **`SecondChangeHandler`** prosleđujući metod za obradu događaja **`TimeHasChanged`**.
- Posle toga, klasa **`DisplayClock`** povezuje taj delegat sa **DOGAĐAJEM `OnSecondChange`** klase **`Clock`**.

# Pretplata na korisnički događaj* (2)

```
public class DisplayClock
{
    public void Subscribe(Clock theClock)
      {
        theClock.OnSecondChange +=
                  new Clock.SecondChangeHandler(TimeHasChanged);
       }
  public void TimeHasChanged(object theClock, TimeInfoEventArgs ti)
    {
          Console.WriteLine("Current Time: {0}:{1}:{2}",
                                  ti.hour.ToString(), ti.minute.ToString(),
                                                  ti.second.ToString());
    }
}
```

Klasa DisplayClock koja se pretplaćuje na događaj **OnSecondChange**

Pretplaćivanje na događaj **OnSecondChange**

Metoda za **obradu događaja**

# Pretplata na korisnički događaj* (3)

```
public class LogCurrentTime
 {
  public void Subscribe(Clock theClock)
   {
     theClock.OnSecondChange +=
                new Clock.SecondChangeHandler(WriteLogEntry);
    }
public void WriteLogEntry(object theClock, TimeInfoEventArgs ti)
 {
   Console.WriteLine("Logging to file: {0}:{1}:{2}",
     ti.hour.ToString( ), ti.minute.ToString( ), ti.second.ToString( ));
 }
}
```

Klasa **LogCurrentTime** koja takođe reaguje na događaj **OnSecondChange**

Metoda za pretplaćivanje na događaj **SecondChange**

Metoda za obradu događaja

# Testiranje delegata*

```
public class Test
 {
     public static void Main()
   {
     Clock theClock = new Clock();
     DisplayClock dc = new DisplayClock();
     dc.Subscribe(theClock);

     LogCurrentTime lct = new LogCurrentTime();
     lct.Subscribe(theClock);

  theClock.Run();
  }
 }
}
```

Novi objekt klase **Clock**

Objekt **DispalyClock** i **pretplata** na napravljeni objekt **theClock**

Objekt **LogCurrentTime** i pretplata na napravljeni objekt **theClock**

Startovanje časovnika