

# *Kodiranje GUI aplikacija u Visual C#*



# UVOD

## *Uvod*

*'Kodiranje grafičke aplikacije u Visual C#:*

*Ova lekcija se bavi događajima, programiranjem događaja. Daćemo jedan primer event-driven-programming-a, u vidu „mašine za sabiranje“. Objasnićemo takodje šta je to Event Handler i Solution Explorer. Isto tako, pomenućemo kako se otklanjaju sintaksne greške, i šta je to Online-help.*

*Ključna pitanja:*

*Šta su to događaji u Visual C#?*

*Kako se programiraju događaji?*

*Šta je to Solution Explorer i kako se koristi?*

# Postavljanje objekta “dugme” na C#-formu

---

01

# OBJEKAT DUGME

*Ako se želi postaviti tekst na Button, onda se u Properties window-u pod stavkom Text otkuca npr. tekst **Change the label** ili **Promeni tekst** (umesto teksta *button1*).*

Objekat Dugme, tj. *Button*, je jedan od najkorisnijih objekata u Visual C#, jer pomoću objekta Dugme može se korisnik upravljati nekom aplikacijom tj. programom. Naime, koristeći taj objekat u Visual C#, korisnik programa tj. aplikacije određuje šta će se desiti u aplikaciji posle aktiviranja tog objekta (tj. kada klikne *Button* sa mišom).

Sledeći su koraci da se postavi Dugme na formu,

Da bi se postavio objekat *Button* na nekoj Formi, a već smo definisali ranije šta je Forma i kako se kreira, treba kliknuti ikonu pored objekta *Button* u *Toolbox*-u.

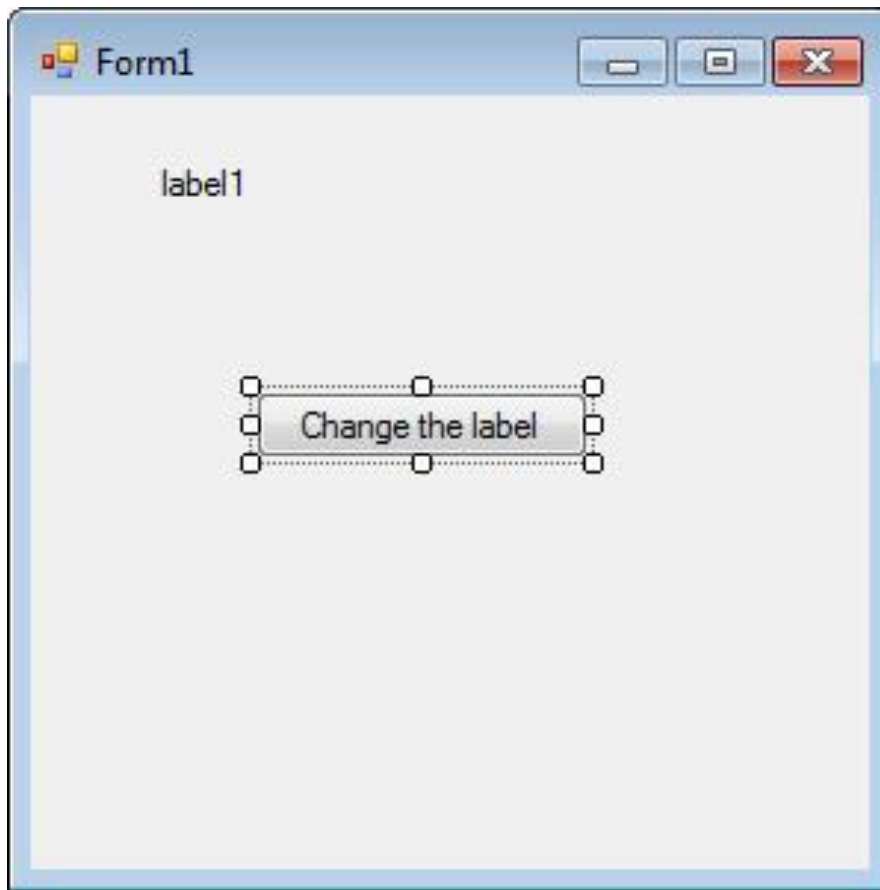
Kliknuti na Formu, onda vući pokazivač tj. strelicu miša nadole i nadesno ,na dno Forme (*drag down and right*), i onda osloboditi dugme miša, i onda će se pojaviti pravougaonik koji označava Dugme tj. *Button*. Taj pravougaonik se može pomerati ako se klikne na sredini pravougaonika koji označava *Button*, držanjem levog dumeta miša i povlačenjem objekta *Button*. Takodje, može se menjati veličina objekta pomoću dugmića po ivicama objekta *Button*.

Ako se želi postaviti tekst na Button, onda se u *Properties window-u* pod stavkom Text otkuca npr. tekst **Change the label** ili **Promeni tekst** (umesto teksta *button1*).

Dole je prikazana jedna Forma sa Dugmetom i Nalepnicom (objekat Label) , gde je tekst Dugmeta na Formi: **Change the label** , a tekst na Nalepnici je : **label1**.

# C# - FORMA SA DUGMETOM "CHANGE THE LABEL"

*Na slici je C# - forma sa Dugmetom "Change the label" .*



Slika 1: forma sa dugmetom

# Kodirajući editor

---

02

# KODIRAJUĆI EDITOR

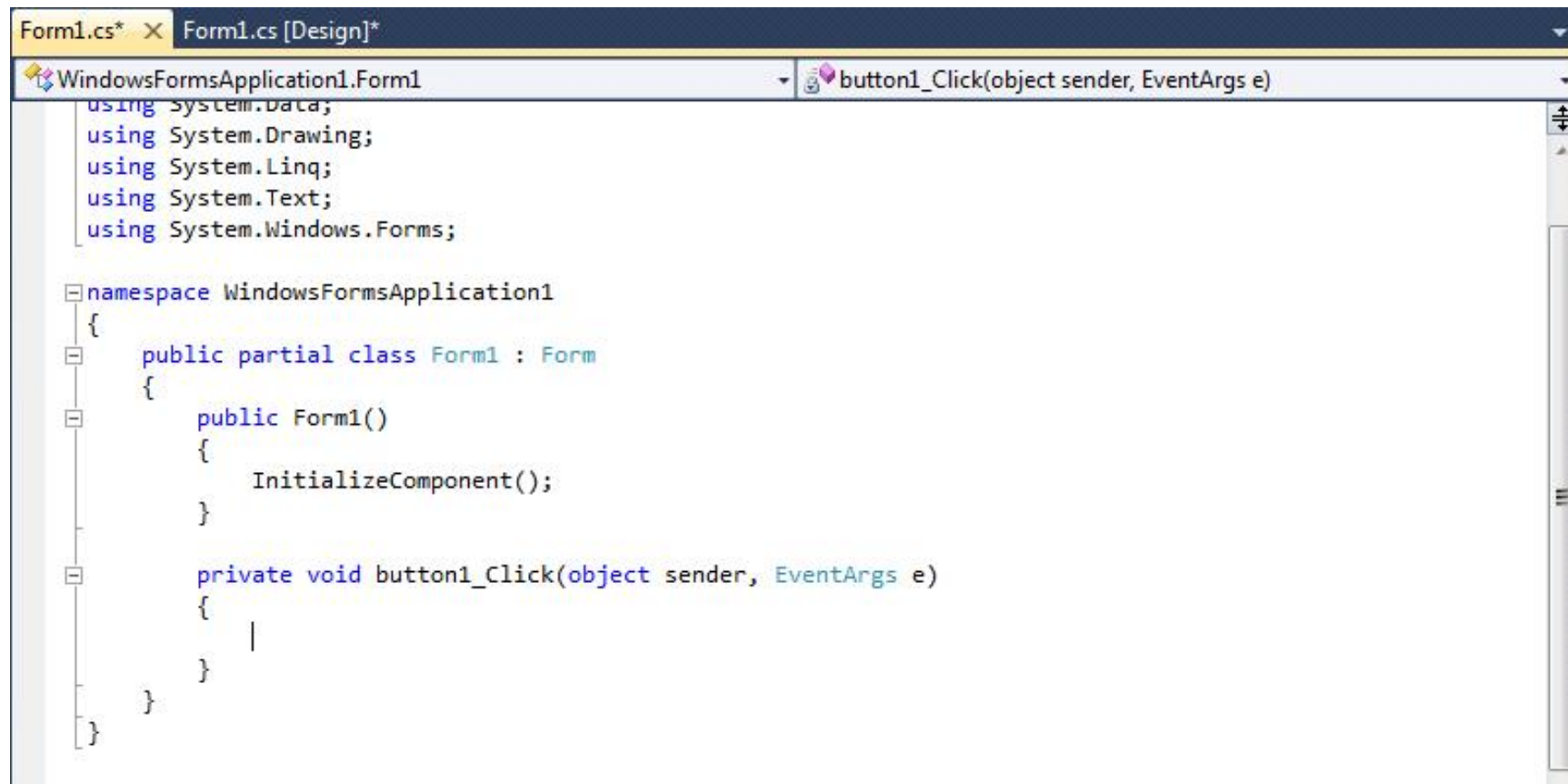
*Kada se otvori kodirajući editor, neke instrukcije su automatski generisane od strane Visual Studio IDE.*

Posmatrajmo primer gde je jedna C#-forma (tj. Windows-forma) postavljena u centralnoj radnoj površini Visual Studio-a, i na toj formi je postavljeno jedan objekat **Button1**. Ako se pritisne dugme Form1.cs\* umesto Form1.cs (Design)\* (na glavnoj strani Visual Studio-a je „central work area“ na čijem vrhu se nalaze ova dugmad), na centralnoj radnoj površini otvoriće se kodirajući editor (**Code Editor**) koji služi za ukucavanje programskih instrukcija. Kadase otvori kodirajući editor, neke instrukcije su automatski generisane od strane **Visual Studio IDE**, a neke je potrebno da ukuca programer. Pogodnost Visual Studio-a je da automatski generiše deo programa da bi olakšalo programerski posao. Automatski generisani kod se generiše pomoću biblioteke .NET.

Na donjoj slici je prikazan kodirajući editor sa programskim instrukcijama koje su automatski generisane, a programer ukucava instrukcije izmedju dve velike zagrade (instrukcije koje se odnose na objekat Button1).

# KODIRAJĆI EDITOR

*Slika prikazuje kodirajći editor, i zagrade { } izmedju kojih programer ukucava programske instrukcije.*



```
Form1.cs* X Form1.cs [Design]*
WindowsFormsApplication1.Form1 button1_Click(object sender, EventArgs e)
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();
        }

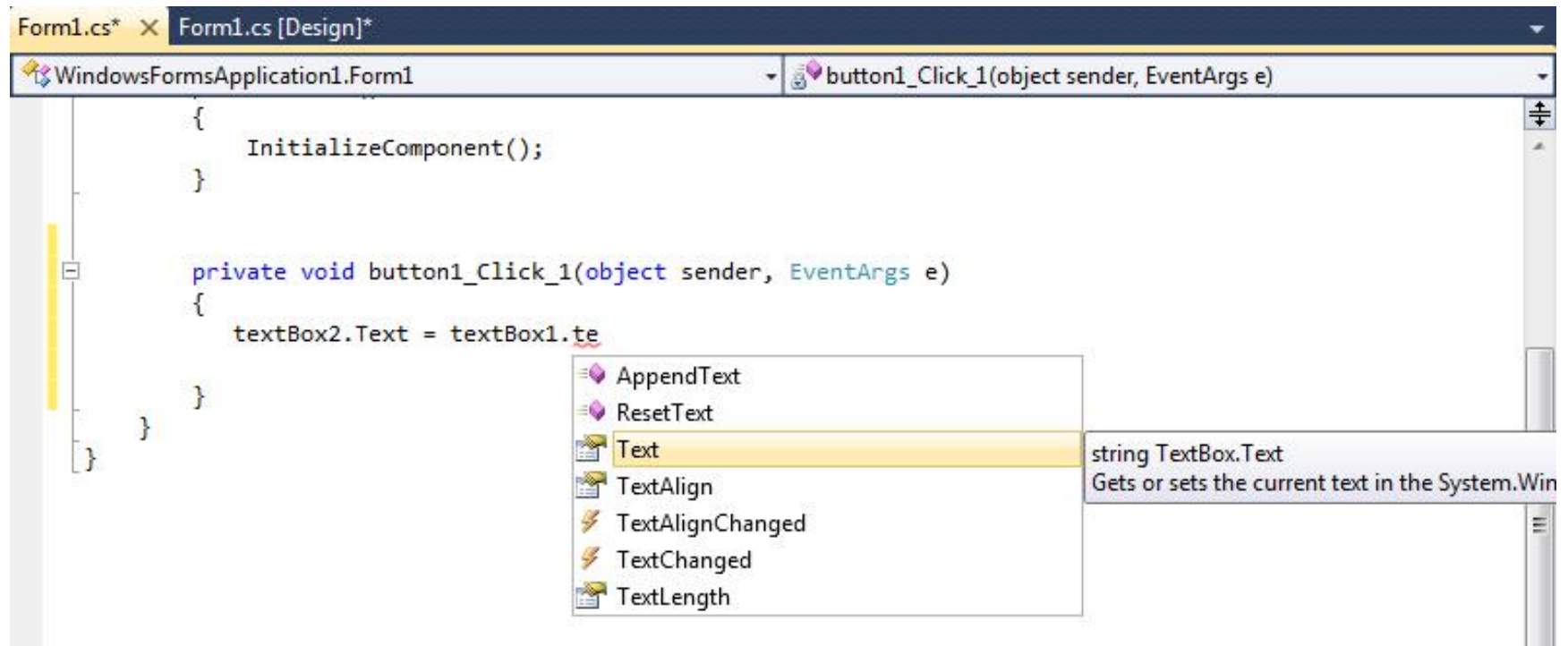
        private void button1_Click(object sender, EventArgs e)
        {
            |
        }
    }
}
```

Sl.1: Code-editor



# KODIRAJUĆI EDITOR

*Slika ilustruje rad sa kodirajućim editorom.*



Sl.2: Code-editor

# Događaj (Event) u Visual C#

---

03

# DOGAĐAJ

*Događaj, tj Event, to je neka akcija korisnika.*

Većina Windows aplikacija radi tako da odgovara na akcije korisnika. Npr. kada pritisnete dugme na tastaturi ako ste u *word processor*-u, neki znak se pojavljuje u dokumentu. Neki događaj, tj. *event*, npr. pritiskanje dugmeta, je praćen odgovorom tj. reakcijom programa. Ovakav proces tj. koncept odgovaranja programa tj. aplikacije na događaje, je vitalan za kreiranje Visual C# aplikacija. Naime, vrši se pisanje instrukcija u programu tj. kodiranje aplikacije, i ove instrukcije tj. neki blok programskog koda, će biti izvršen kada se desi neki događaj. Odgovor na događaj tj. akciju korisnika je izuzetno važan za kreiranje programa tj. aplikacija u Visual C#. Odgovor se sastoji u izvršavanju instrukcija od strane programa, onda kada se neki događaj desi. Ovo je tzv. **Event-driven programming**.

Dakle, događaj, tj *Event*, to je neka akcija korisnika. A klasični primer događaja je kliktanje dugmeta na ekranu, gde se kliktanje obavlja pomoću računarskog miša. Da bi ilustrovali kako funkcioniše ovaj koncept odgovaranja na događaje, postavimo „dugme“ na Visual C# formu. Da bi demonstrirali programiranje događaja tj. kako se instrukcije zadaju nekoj grafičkoj aplikaciji, možemo posmatrati primer iz prethodne glave, gde smo postavili *Button* na Visual C# formu i nalepnicu. Onda, sa mišem treba duplo kliknuti, *double-click*, na pravougaoniku koji označava *Button*. Rezultat toga je da Visual C# otvara tekstualni editor, tj. editor za kodiranje (*Coding editor*) odnosno za pisanje programskih instrukcija, gde je neki tekst već unet, i ovo izgleda ovako :

```
private void button1_Click(object sender, System.EventArgs e)
{.....}
}
```

I onda se instrukcije tj. programski kod ukucava izmedju velikih zagrada { }, dakle zagrada koje se već nalaze na otvorenoj strani Kodirajućeg editora. Na donjoj slici, je ilustrovano kako to izgleda u editoru za kodiranje. Znači, pomoću *double-click* na pravougaoniku koji označava **Button**, otvara se *code-editor*, gde se može onda ukucavati programski kod.

# KODIRAJĆI EDITOR, I ZAGRADE { } IZMEDJU KOJIH SE UKUCAVA PROGRAMSKI KOD

*Slika prikazuje kodirajći editor, i zagrade { } izmedju kojih se ukucava programski kod.*



```
Form1.cs* x Form1.cs [Design]*
WindowsFormsApplication1.Form1 button1_Click(object sender, EventArgs e)
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)
        {
            |
        }
    }
}
```

# Prvo kodiranje u Visual C# ( programiranje događaja)

---

04

# PROGRAMIRANJE DOGAĐAJA

*Ovde nam je cilj da demonstriramo programiranje događaja. Funkcija koja odgovara nekom događaju se zove **Event handler**.*

Da bi se uradio ovaj primer, i demonstriralo jednostavno kodiranje, korišćićemo jednu C# -formu (Windows formu) sa jednim dugmetom i jednom nalepnicom, tj. primer iz prethodne glave. Zatim, treba uraditi sledeće korake:

1. Otvoriti *code-editor* (tekst editor) za klik-događaj, tako što se duplo klikne *Button* u Forma-dizajneru.
2. Otkucati u tekst editoru : `label1.t`, i onda se pojavljuje padajući-meni, *drop-down menu*, a zatim izabrati `Text` u tom meniju, onda pritisnuti *Spacebar* ( tj. dugme tastature koje pravi prazno mesto u tekstu), i pojavljuje se automatski reč `Text` tj. `Label1.Text`:

```
private void button1_Click(object sender, System.EventArgs e)
```

```
{label1.Text
```

```
}
```

3. Kompletirati liniju koda tako da se otkuca: `Label1.Text = "PROMENA"`; i pri tome ne zaboraviti znak tačka-zarez. Dobijamo:

```
private void button1_Click(object sender, System.EventArgs e)
```

```
{label1.Text = "PROMENA";
```

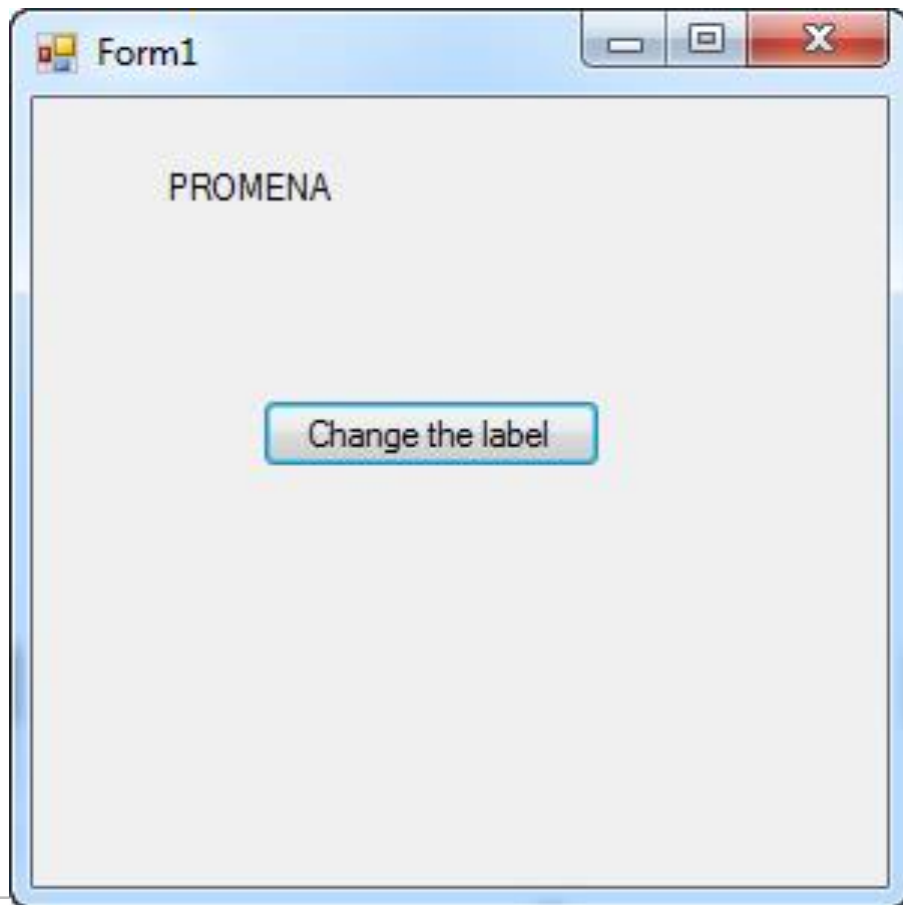
```
}
```

4. Izvršiti aplikaciju tako što se klikne `Start-button` (mala crna desna strelica ispred `Debug`) na `Visual Studio Toolbar`, zatim kliknuti `Button` gde piše **Change the label**, i kao rezultat tekst nalepnice se onda menja i postaje: **PROMENA**, što je prikazano na donjoj slici.

**Event handler:** Napomenimo, da funkcija koja odgovara nekom događaju se zove *Event handler*. Dakle, kod programiranja aplikacija, *Event handler* je programski kod koji treba napisati koji će se izvršiti kada neki događaj (*event*) se desi.

## FORMA SA DUGMETOM " CHANGE THE LABEL "

*Na slici se vidi forma sa Dugmetom " **Change the label**" i promenjenim tekstom u Nalepnici.*



# PROGRAMIRANJE OSOBINA OBJEKATA

*Prednost zadavanja osobina objekta pomoću programskog koda a ne Property editor-a je što se to može raditi u toku izvršavanja programa kliktanjem dugmeta.*

## Zadavanje osobina objekata pomoću programskog koda (Programiranje osobina objekata):

Programski kod koji smo napisali prethodno je bio samo jedna linija tj. jedna instrukcija, i to

label1.Text = "PROMENA"; Dakle, omogućeno je da se promeni naslov Nalepnice pomoću objekta dugme (kliktanjem dugmeta).

Ali, kreiranje Nalepnice sa naslovom PROMENA, se umesto kodiranjem u tekst editoru može postići pomoću *Property editor-a*. Naime, treba uraditi sledeće:

1. Izabere se „label” u **Toolbox-u**,
2. I onda u *Properties window* u *Text property* se ukuca: PROMENA

Ova instrukcija ima efekat zadavanja osobine Text property za objekat nazvan „label1” da ima vrednost „PROMENA”.

Medjutim, prednost zadavanja osobina objekta pomoću programskog koda a ne **Property editor-a** je što se to može raditi u toku izvršavanja programa kliktanjem dugmeta, umesto što se to uradi unapred pre izvršavanja programa. I druge osobine objekta, npr.:

label1.Font ili label1.**BackColor**,

mogu se promeniti na isti način, pomću kodiranja u tekst editoru, kao što je to uradjeno sa label1.Text. I ove osobine objekta se ostvare posle kliktanja dugmeta „**Change the label**”.



# LISTA DOGAĐAJA

*Ako se duplo-klikne u prostoru sa desne strane pored imena događaja, pojavljuje se u tekst editor kostur programskog koda koji će se izvršavati kada se desi taj događaj.*

Objekat Button reaguje na kliktanje tj. događaj kliktanja. Medjutim objekat Button može da reaguje na čitavu listu događaja. Naime, ako se izabere objekat **Button** u Properties window, i onda ako se pritisne *Events icon*, tj. Ikonica koja izgleda kao neka munja (*lightning flash*), onda se u Properties window pojavljuje lista događaja, npr.:

....

BackColorChanged

BackgroundImageChanged

.....

Click

.....

DragDrop

DragOver

.....

Pri tome, ako se duplo-klikne u prostoru sa desne strane pored imena događaja, npr. pored događaja **Click**, pojavljuje se u tekst editor kostur programskog koda koji će se izvršavati kada se desi taj događaj. Npr programski kod u prethodnoj glavi se može uneti na taj način. Dakle, ako se izabere događaj Click za objekat Button, može se ukucati.

```
label1.Text = "PROMENA";
```

# Primeri programiranja dogadaja

---

05

# PRIMER

*Ovaj primer opisuje kako da se pomoću Visual C# napravi tj. izprogramira kalkulator.*

Ovaj primer opisuje kako da se pomoću Visual C# napravi tj. izprogramira „računska mašina“ tj. kalkulator za sabiranje dva broja, i prikazivanje rezultata tog sabiranja. Programiranje Mašine za sabiranje izgleda ovako:

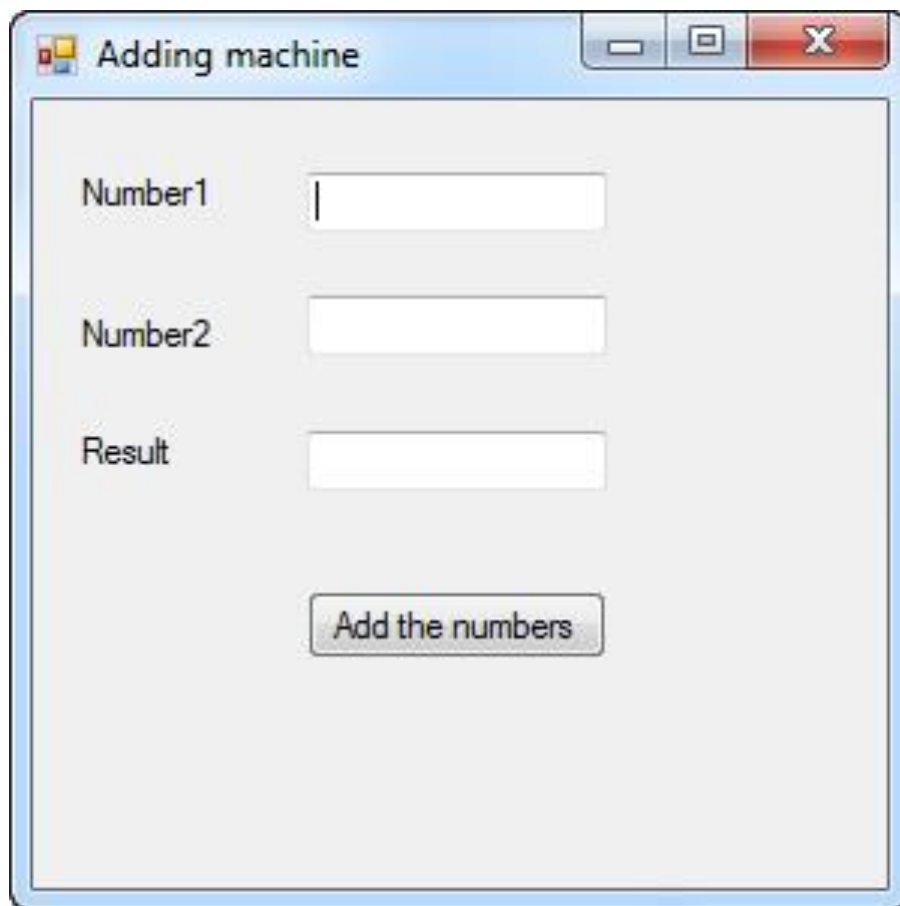
1. U **Toolbox window-u** naći objekat **TextBox**, i dodati dva takva objekta na Formu (kojoj se da ime *Adding machine*), pri tome zadaje se Text property da je prazno (nema ništa), za ova dva objekta.
2. Dodati 4 Nalepnice, tj. 4 puta postaviti Label na Formu, i jedan put postaviti **Button**, kao na slici dole.
3. Zadati ime za svaku od četiri Label koje su postavljene na Formu, i to pomoću Properties window i **Name property**, npr. za Label nazvanu *Result here* se **Name property** zada kao *lbResult*
4. Napisati samo jednu liniju programskog koda za objekat **Button** i događaj Click:  
**lbResult.Text = (float.Parse(textBox1.Text) + float.Parse(textBox2.Text)).ToString();**
5. Ako se klikne Start button, može se testirati gornja aplikacija. Onda treba ukucati brojeve u dve TextBox, i onda kliknuti dugme da izvrši sabiranje i prikaže rezultat.

Ovo kodiranje gore prikazano, naime ta jedna linija programskog koda, sada izgleda dosta nerazumljivo, ali će biti detaljnije opisano kasnije, u nekom drugom predavanju. Na sličan način može se napraviti „kalkulator“ koji množi dva broja (prikazati C# - formu napravljenu za tu svrhu?).

U ovom primeru smo nalepnici „label4“ zadali novo ime „lbResult“, jer je bolje da neki objekat ima ime koje ima značenje, nego jednostavno „label4“, koje ne ukazuje za šta taj objekat služi.

## „ADDING MACHINE“ NAPRAVLJENA POMOĆU VISUAL C# FORME

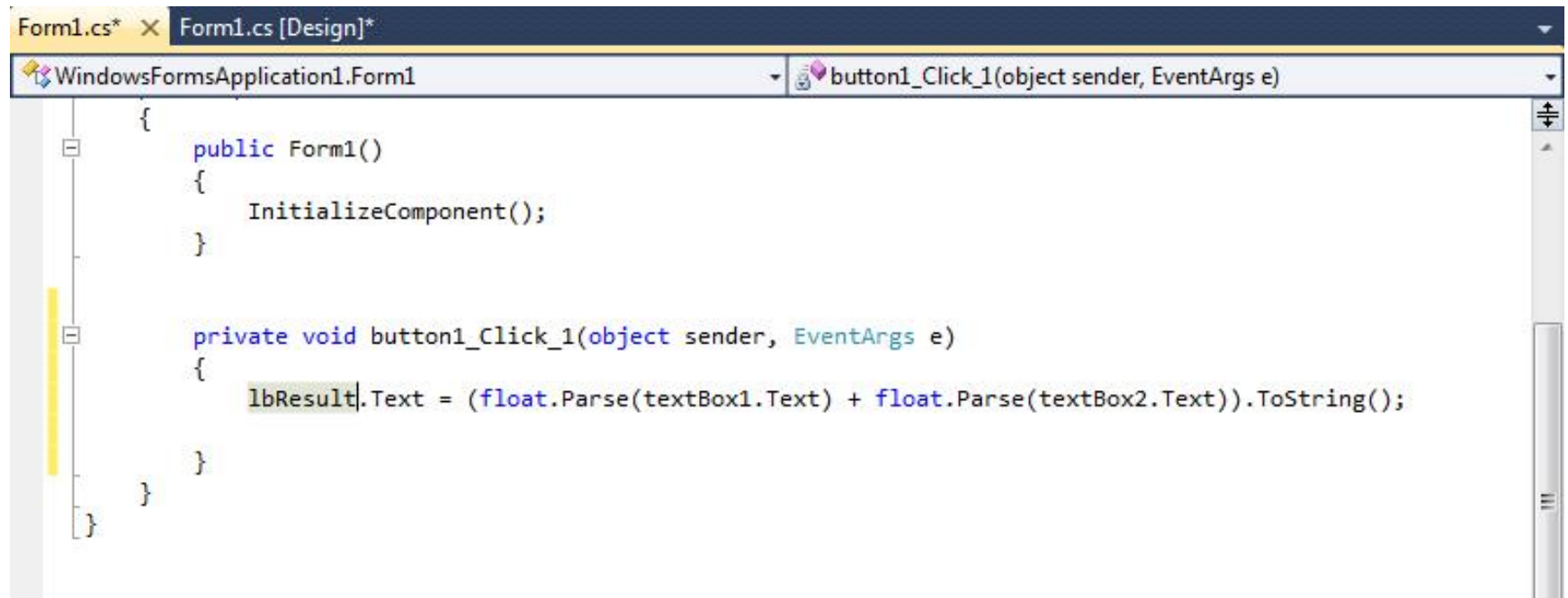
*Na slici je „Adding machine“ napravljena pomoću Visual C# forme.*



The image shows a screenshot of a Windows application window titled "Adding machine". The window has a standard Windows title bar with minimize, maximize, and close buttons. Inside the window, there are three text input fields arranged vertically. The first field is labeled "Number1" and contains a single vertical bar character "|". The second field is labeled "Number2" and is empty. The third field is labeled "Result" and is empty. Below the input fields is a button labeled "Add the numbers".

# KODIRANJE POMOĆU CODE-EDITOR-A

*Slika ilustruje kodiranje pomoću Code-editor-a (tekst editora).*



```
Form1.cs* x Form1.cs [Design]*
WindowsFormsApplication1.Form1 button1_Click_1(object sender, EventArgs e)
{
    public Form1()
    {
        InitializeComponent();
    }

    private void button1_Click_1(object sender, EventArgs e)
    {
        lbResult.Text = (float.Parse(textBox1.Text) + float.Parse(textBox2.Text)).ToString();
    }
}
```

Sl.2: Code-editor

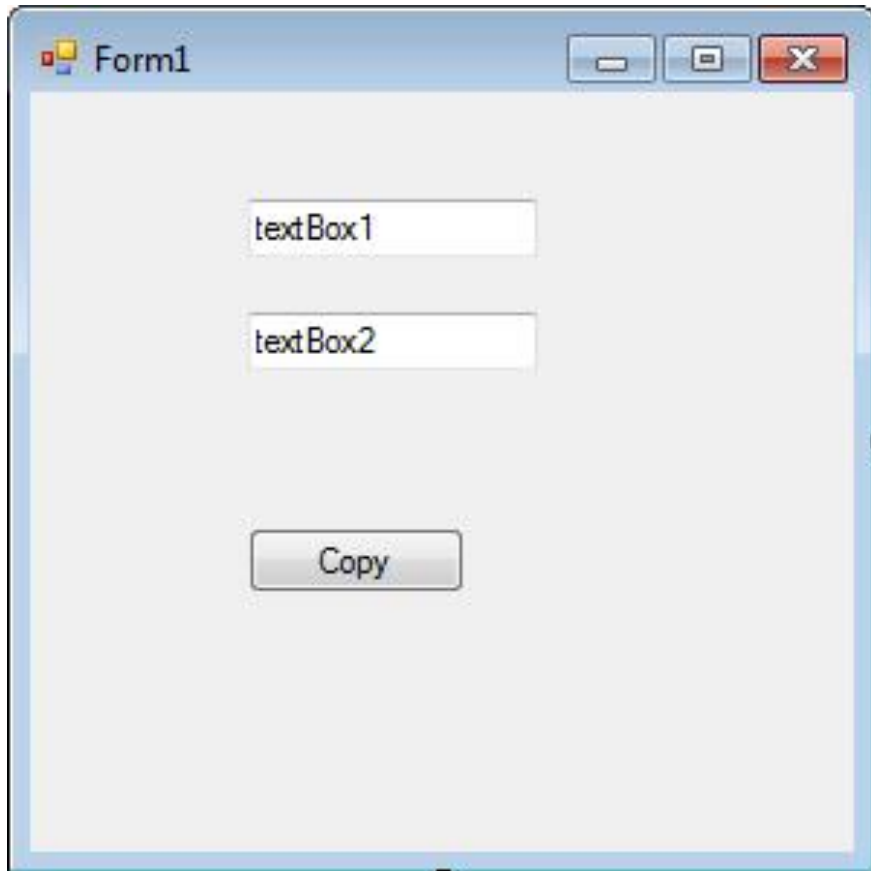
## PRIMER 2

*Na sličan način, može se npr. napraviti „forma“ koja učitava tekst pomoću textBox1 i kopira taj isti tekst u textBox2 po pritiskanju dugmeta button1.*

Takođe, na sličan način, može se npr. napraviti „forma“ koja učitava tekst pomoću textBox1 i kopira taj isti tekst u textBox2 po pritiskanju dugmeta *button1*. To bi bila forma nazvana *Copy App*. Pri tome, je prethodno izmenjen tekst na „kontrolama“ textBox1, textBox2, i button1, i stavljeno je, respektivno, npr. sledeći tekst, „*Text*“, zatim „*Text copy*“, i „*Copy*“. Ovo se radi pomoću prozora *Properties* i *Text property*, pri čemu se ukuca željeni tekst i pritisne Enter. Takođe, slično je pomoću *Text property* „forme“ Form1, zadat tekst „Copy App“ umesto „Form1“. Dalje, potrebno je kliknuti *AcceptButton* property na listi *Form properties*, i onda izabrati koje dugme odgovara na Enter-dugme, i to je dugme *button1*. Konačno, za donji tekstboks (gde piše „Program copies“), izabrati *Read only* na listi *Behaviour properties*. Na kraju je potrebno kliknuti *Save button* u *Visual Studio Toolbar* da bi se memorisale uradjene promene.

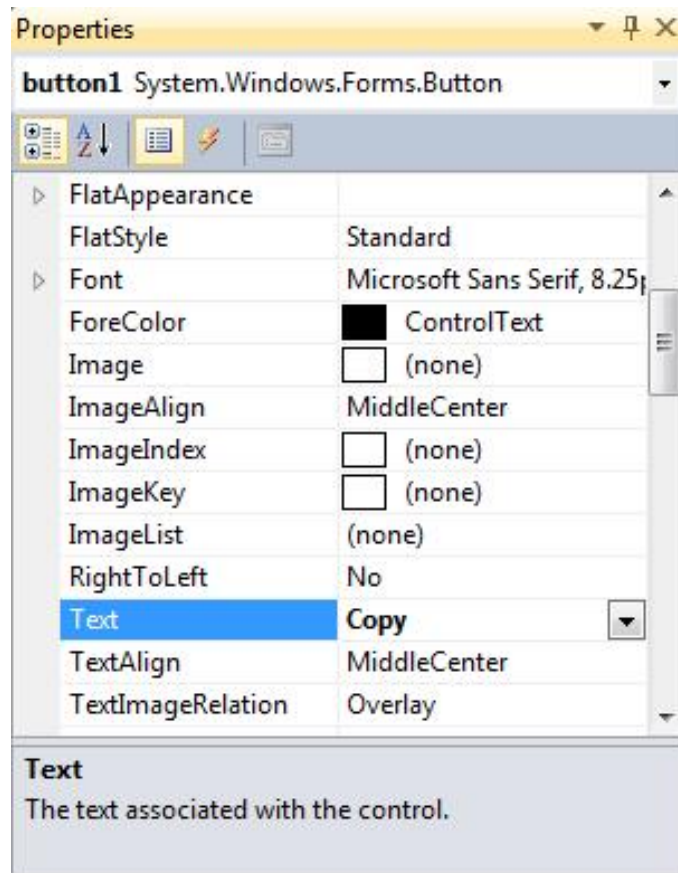
# „FORMA“ FORM1

*Na slici je „Forma“ Form1.*



# PROPERTIES WINDOW ZA OBJEKAT BUTTON1

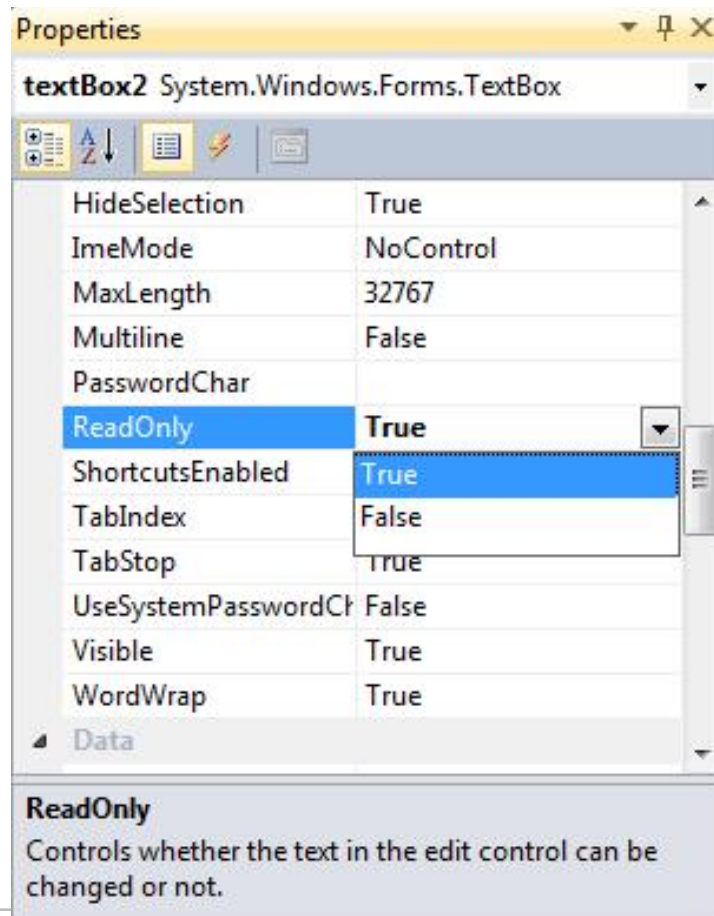
*Slika prikazuje Properties window za objekat button1.*





# PROPERTIES WINDOW ZA OBJEKAT TEXTBOX2

*Slika ilustruje Properties window za objekat textBox2.*



# Greške, i Help

---

# 06

# ERROR MESSAGE

*Šta će se desiti kod izvršavanja programa? Jednostavno, okruženje Visual C# će zaustaviti program, i prikazati tzv. Error message.*

Ako postoje sintaksne greške tj. greške u kucanju programskog koda, onda kod egzekucije programa (tj. aplikacije koja sadrži takve greške), npr. ako se ukuca

`textBox1.Tx` umesto `textBox1.Text`,

(gde ima sintaksna greška) šta će se desiti kod izvršavanja programa?

Jednostavno, okruženje Visual C# će zaustaviti program, i prikazati tzv. *Error message*, npr.

*There were build errors, continue?*

*Yes or No?*

Tada treba odgovoriti sa No. Onda pogledati:

**Task List**,

koja se tada pojavi, i ova lista sadrži listu greški. Naime, pojaviće se lista greški, i ako se duplo-klikne na neku grešku u toj listi greški, onda prikazaće se (u tekst editoru) pogrešna linija programskog koda.

# KODIRAJUĆI EDITOR

*Slika ilustruje rad sa kodirajućim editorom.*

```
{  
lbResult.Text = (float.Parse(textBox1.Tx) + float.Parse(textBox2.Text)).ToString();  
}
```

Sl.1: Code-editor

# ERROR MESSAGE

*Slika prikazuje Error message (poruka da se pojavila greška kod izvršavanja programa).*



Sl.2: Error message

# HELP-MENI

Iz **Help-menija**, izabрати *Contents* da bi se pristupilo online-verziji *Visual C# Manuals*.

S druge strane, korisno je naučiti kako koristiti **Online Help**, jer Visual C# je obiman i složen jezik. **Online Help** dokumenti se omogućuje iz *online MSDN*, *Microsoft Development Network*, odakle se mogu dobiti:

*online help documents*.

Iz **Help-menija**, izabрати

*Contents*

da bi se pristupilo *online-verziji Visual C# Manuals*, kao i članaka. Ako se postavi *cursor* na neku reč oko koje je potrebna pomoć, i pritisne F1, to će da pruži HELP tj. pomoć oko te reči. Naime, *Online help* je integrisano sa MSDN, i ako se pritisne F1, onda ustvari mogu se naći *online MSDN*-članci.

# Solution Explorer Window

---

07

# SOLUTION EXPLORER

*Solution Explorer je WINDOW, tj. PROZOR, koji prikazuje sve elemente u jednom **projektu**.*

Neka aplikacija u Visual C# može da sadrži ne samo jednu FORMU, već nekoliko FORMI. Zatim, može da ne sadrži ni jednu FORMU, već samo jedan ili višemodula čistog programskog koda.

Kod velikih projekata, može biti puno elemenata koji čine **projekt**, i rukovođenje tim elementima nože biti teško. Zato,

## Solution Explorer

je **WINDOW**, tj. PROZOR, koji prikazuje sve elemente u jednom **projektu**. On se nalazi sa desne strane u gornjem delu na stranici Visual C# IDE.

Ako **Solution Explorer** nije vidljiv, može se naći u **Wiew-menu**, ili treba pritisnuti Ctr+Alt+1. I inače, ako se ne vidi Solution Explorer, ili Properties Window, ili Toolbox, ili ostali prozori u Visual C#, mogu se restaurirati tako što se koristi Wiew-meni.

**Solution-Explorer-Window** ima u gornjem levom uglu tri ikonice, i to, prva levo ikonica omogućuje da se vidi kod-editor za selektovani objekt, druga levo ikonica, da se vidi sam objekat, npr. FORMA, i treća ikonica s levo, omogućuje da se vide **File-Properties** za selektovani objekt na listi.

Znači, **Solution Explorer** se koristi ako želite da vidite C# -formu (tj. formu kako je definisana u Visual C#), ili modul programskog koda. Ovo je vrlo dragoceno kod **velikih projekata**.



# SAVING YOUR PROJECT (MEMORISANJE PROJEKTA):

*Ako se izabere **Save iz File-menu**, samo onaj fajl koji je trenutno aktivan se memoriše na taj način.*

Neki Visual-C#-projekt tj. aplikacija tj. program sastoji se od serije fajlova memorisanih na disku. Npr. najjednostavniji projekt koji pravi jednu C# -formu , sadrži

Project-file, sa ekstenzijom **.csproj**,

Form-file, sa ekstenzijom **.cs**,

I neke druge propratne fajlove.

Ako se izabere **Save iz File-menu**, samo onaj fajl koji je trenutno aktivan se memoriše na taj način.

A da bi se memorisao ceo projekt, treba izabrati:

**Save All**

A ako hoćete da napustite okruženje Visual C#, zahtevaće se memorisanje fajlova koji nisu memorisani. S druge strane, fajlovi se memorišu automatski ako se vrši egzekucija tj. izvršavanje projekta.

Ime fajla se određuje kada se dodaje novi element, C#-forma ili kod-modul. Imena se mogu preimenovati pomo'u komande RENAME, koja se pojavi na list posle desnog-klika.

# REOPENING A PROJECT (PONOVRNO OTVARANJE PROJEKTA):

*Može se otvoriti više projekata istovremeno.*

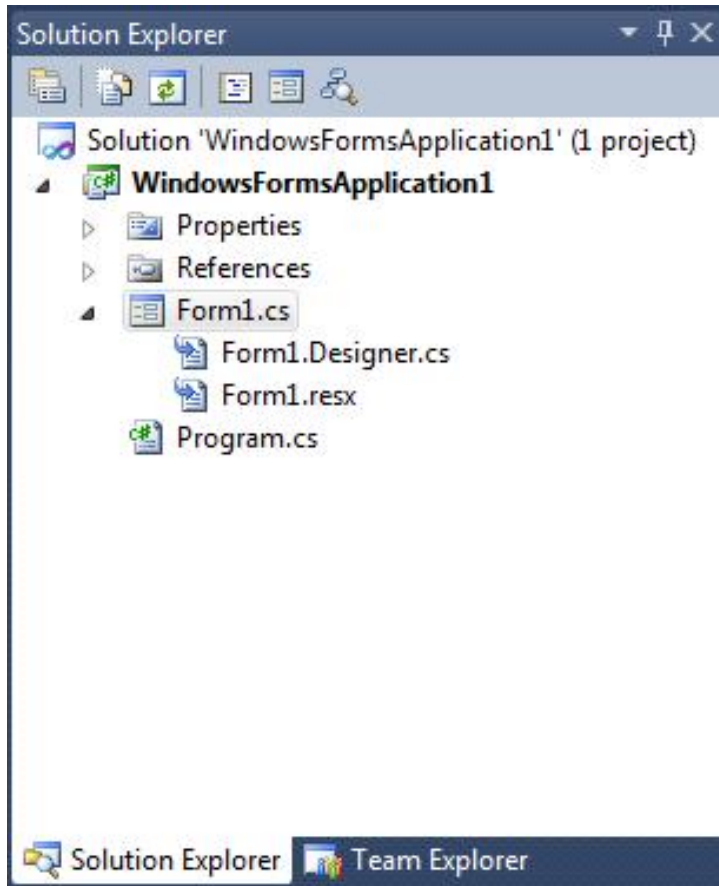
Da bi se ponovo otvorio neki Visual-C#-projekt, treba uraditi

1. Izabrati **Open>Project** iz **File menu**
2. Naći projektni fajl sa ekstenzijom **.csproj** ili solution-fajl sa ekstenzijom **.sln** i kliknuti na tom fajlu, da bi se on selektovao,
3. Kliknuti **Open** na prozoru **Open Project** u donjem desnom uglu

Može se otvoriti više projekata istovremeno. A Solution je u Visual C# definisano kao kontejner za nekoliko projekata.

# SOLUTION EXPLORER WINDOW

*Slika ilustruje Solution Explorer Window.*



# Višestruke forme

---

08

# VIŠESTRUKÉ FORME

*U praksi se često koriste aplikacije koje koriste višestruke Windows forme.*

Do sada smo posmatrali aplikacije koje su koristile samo jednu „formu“ (C# - Windows formu). Međutim, u praksi se često koriste aplikacije koje koriste višestruke Windows forme. Evo kako možemo da dodamo dodatnu Windows formu nekom projektu.

1. Postaviti dugme **button1** na formu (**Form1**).

2. Zabrati **File menu > Add new item**.

3. Zatim izabrati Windows form, i prihvatiti predefinisano (*default*) ime **Form2.cs**.

4. Nova forma je u stvari klasa, i treba ukucati sledeći programski kod za *click event handler* za dugme **button1** na formi **1**, gde se kreira objekt te klase, i zatim poziva metoda Show():

```
private void button1_click(object sender, System.EventArgs e)
{
    Form2 f = new Form2 ();
    f.Show();
    f.Dispose();
}
```

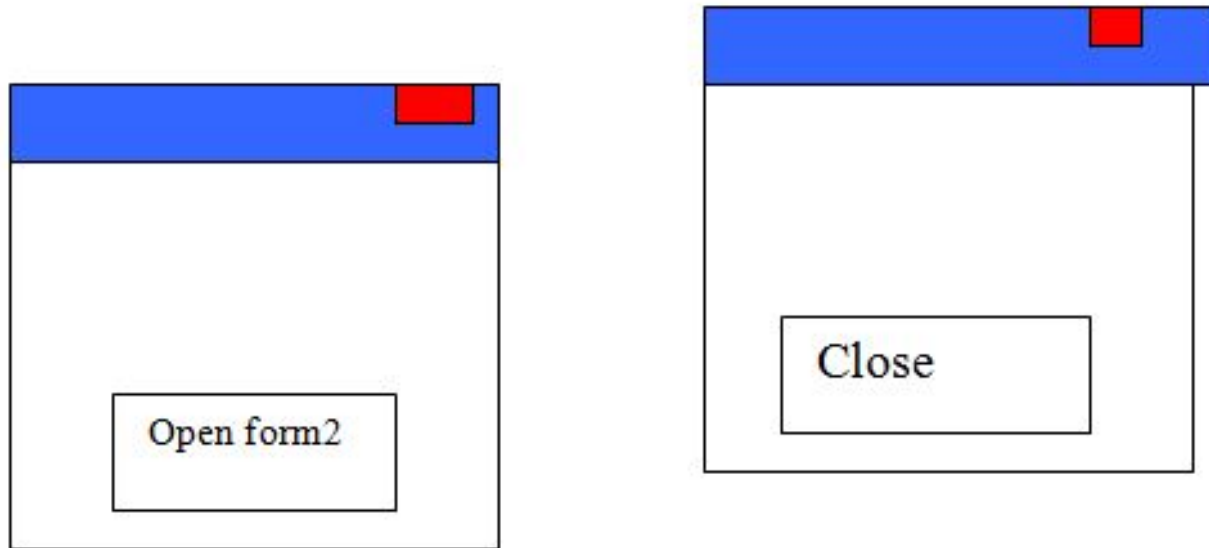
Dakle, za **button1** na formi **1**, kreirali smo prethodni klik *event handler*.

5. Na formu **2** može se postaviti dugme na kome piše *Close*, i za klik *event handler* za to dugme može se ubaciti metoda Close() koja zatvara formu.

6. Izvršiti aplikaciju, pritisnuti dugme na prvoj formi, i pojavljuje se forma **2**. Druga forma može da se zatvori ili klikanjem crvenog dugmeta gore desno na formi, ili klikanjem postavljenog dugmeta na toj formi. Prva forma može da se zatvori klikanjem crvenog dugmeta na prvoj formi.

# VIŠESTRUKÉ FORME

*Slika prikazuje višestruke forme.*



Sl.1: višestruke forme

# MDI

*Ako se ode na Tools meni, pa Options, u Options dialog-u može se izabrati MDI environment umesto Tabbed documents.*

Aplikacije sa višestrukim formama mogu biti nezgodne za rad, i u takvim slučajevima MDI tehnika je korisna da se primeni. MDI to je skraćenica od **Multiple Document Interface**. To je tehnika pomoću koje se rukuje višestrukim Windows formama tj. višestrukim prozorima, tako što se višestruki prozori zadržavaju u okviru glavnog prozora. Ovaj glavni prozor se zove MDI *master form*, a ostali prozori se zovu *MDI-children*, i ovi *MDI-children* mogu se pomerati samo u okviru *MDI master form*. MDI se koristi npr. za *word procesore*. Ovde nemamo vremena i prostora da se dalje bavimo sa MDI. Ako se ode na *Tools meni*, pa *Options*, u *Options dialog-u* može se izabrati *MDI environment* umesto *Tabbed documents*. Kod MDI, može se istovremeno raditi sa nekoliko formi, gde nekoliko formi se nalazi u centralnom prostoru ekrana, i svaki pojedinačni prozor može da se minimizira ili maksimizira ili prikaže kaskadno.

# Vežba 2

---

09



# DIZAJN APLIKACIJE KALKULATOR

*Aplikacija kalkulator treba da sadrži sve elemente osnovnih matematičkih operacija.*

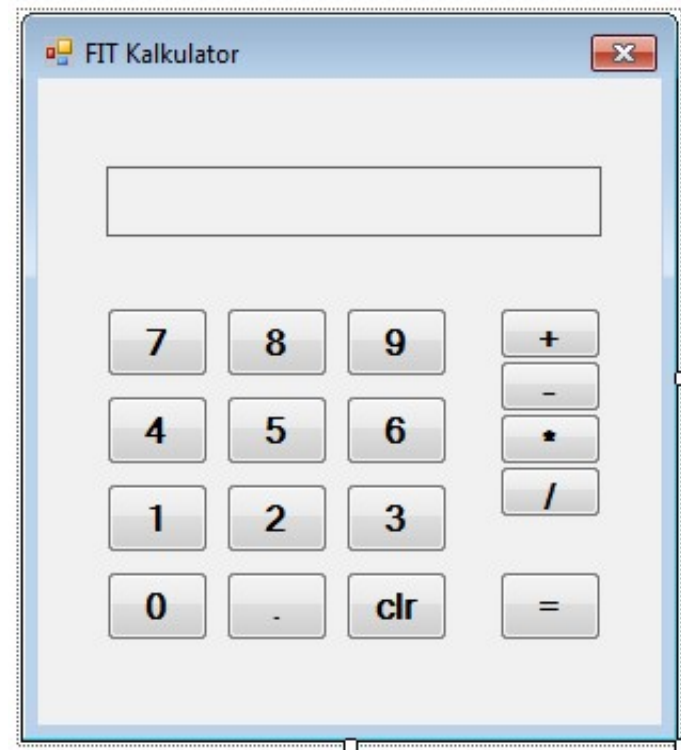
Aplikacija kalkulator treba da sadrži sve elemente osnovnih matematičkih operacija. Shodno tome, moramo dodati samoj aplikaciji dugmad za brojeve, dugmad za računске operacije, dugme za rezultat i tekst box za prikaz rezultata (ekran).

Sve ove elemente potrebno je složiti u pravilnom redosledu, kako bi kalkulator predstavljao stvarni kalkulator i rad na njemu bio intuitivan i lak.

Za vašu vežbu, možete slobodno promeniti raspored dugmića i ekrana i postaviti neuobičajene kombinacije.

Dizajn forme sa kontrolama bi načelno trebao da izgleda ovako:

Slika:



Slika-1: Izgled aplikacije kalkulator

# KODIRANJE APLIKACIJE KALKULATOR

*Aplikacija kalkulator treba da sadrži sve elemente osnovnih matematičkih operacija.*

Form1 designer klasa:

```
namespace NovKalkulator
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components
= null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed
resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not
modify
editor.
        /// the contents of this method with the code
        /// </summary>
        private void InitializeComponent()
```

Klasa Form1:

```
namespace NovKalkulator
{
    public partial class Form1 : Form
    {
        private string vrednost;
        private ArrayList cuvanje;
        public Form1()
        {
            InitializeComponent();
            vrednost = "";
            cuvanje = new ArrayList();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

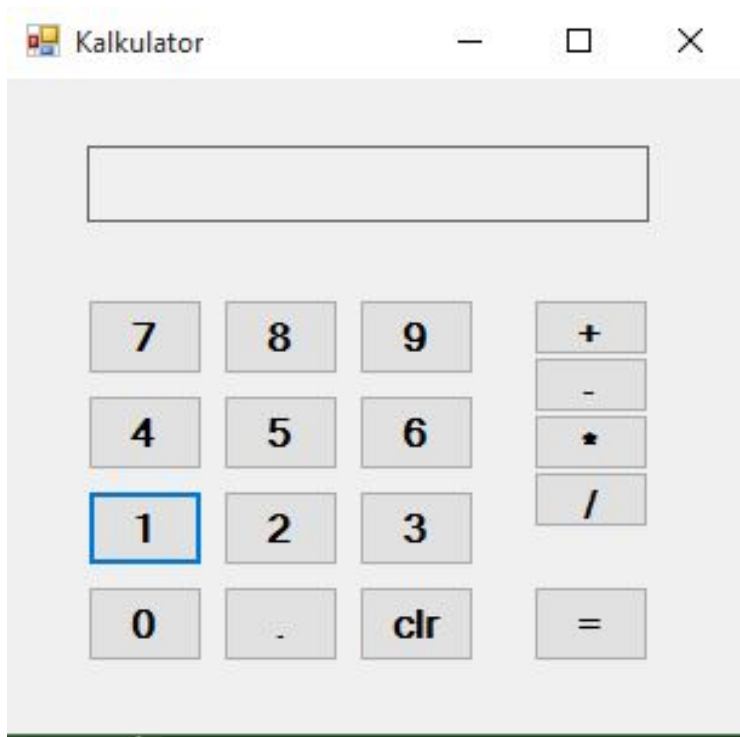
        }

        private void AddOperatorToArray(Button btn)
        {
            cuvanje.Add(vrednost); lblRezultat.Text +=
btn.Text; vrednost = "";
            cuvanje.Add(btn.Text);
            btn.DecimalnaTacka.Enabled = true;
            SetEnableOperatorBtns(false);
        }
        private void AddToArray(Button btn)
        {
            vrednost += btn.Text; lblRezultat.Text +=
btn.Text; SetEnableOperatorBtns(true);
        }
    }
}
```

# IZGLED APLIKACIJE KALKULATOR

*Aplikacija kalkulator treba da sadrži sve elemente osnovnih matematičkih operacija.*

Nakon startovanja aplikacije kalkulator, aplikacija očekuje unos preko događaja miša. Izgled aplikacije je sledeći:



Slika-2: Izgled aplikacije kalkulator nakon pokretanja

# SPAJANJE DVE FORME U C#-U

## *Spajanje dve forme u C#-u*

Kako bi spojili dve forme u C#-u prvo je potrebno da se kreira nova forma. Iz Solution Explorera potrebno je ići desnim klikom na projekat a potom na opciju Add Form. Kada se forma doda i izradi njena funkcionalnost moguće je na prvoj formi napraviti dugme i kreirati klik događaj koji će otvoriti drugu formu.

Kod za otvaranje nove forme je:

```
Form2 forma2 = new Form2();
```

Pored prikazivanje forme putem metode Show moguće je prikazati formu i pomoću metode ShowDialog ali onda će forma biti vezana za prvu formu i prva forma neće moći da se koristi dok je druga prikazana.

Ukoliko želite da prosledite neki parametar do druge forme to je moguće uraditi kroz konstruktor forme.

```
Form2 forma2 = new Form2("Prosledjen naziv");
```

# ZADACI ZA SAMOSTALAN RAD

## Zadatak 1.

Napraviti C# GUI aplikaciju koja ima 4 polja za unos (ime, prezime, indeks, jmbg) , napraviti dugme u donjem desnom delu ekrana i napraviti akciju da kada korisnik klikne na dugme program spoji unete podatke sa zarezima i prikaže kroz MessageBox.

## Zadatak 2.

Napraviti C# GUI aplikaciju u kojoj se upisuju 4 imena klikom na dugme u donjem desnom delu ekrana treba korisniku da se prikaže ukoliko postoji ime koje kreće na slovo A ili najveće uneto ime.

## Zadatak 3.

Napraviti C# GUI aplikaciju koja upisuje unete podatke (ime, prezime, jmbg) u fajl koristeći StreamWriter. Više o StreamWriteru pročitati na internetu.

# Zaključak

---

# ZAKLJUČAK

## *Zaključak*

- Događaj, tj Event, to je neka akcija korisnika, npr. vrlo čest Događaj je kliktanje Dugmeta.
- Odgovor na događaje tj. akcije korisnika je izuzetno važan za kreiranje programa tj. Aplikacija u Visual C#. Odgovor se sastoji u pisanju instrukcija od strane korisnika, onda kada neki Događaj se desi.
- Ako postoje sintaksne greške tj. greške u kucanju programskog koda, onda kod egzekucije programa tj. aplikacije koja sadrži takve greške, npr. ako se ukuca `textBox1.Tx` umesto `textBox1.Text`, šta će se desiti kod izvršavanja programa? Jednostavno, okruženje Visual C# će zaustaviti program, i prikazati tzv. **Error message**.
- S druge strane, korisno je naučiti kako koristiti **Online Help**, jer Visual C# je obiman i složen jezik. Online Help je obično pomoću MSDN, **Microsoft Development Network**, odakle se mogu dobiti *online help documents*. Iz **Help-menija**, izabрати Contents da bi se pristupilo online-verziji **Visual C#.Manuels**.