C# 04. Interfejs IEnumerable, delegati, dogadjaji, Windows forme

Prof. dr Suzana Stojković

Mr Martin Jovanović

Dipl. inž. Ivica Marković

Dipl. Inž. Teodora Đorđević

Sadržaj

- IEnumerator i IEnumerable, ključna reč yield
- Delegati i događaji
- Strukture DateTime i TimeSpan
- Windows forme

Interfejsi IEnumerator i IEnumerable

- Služe za obilazak kolekcije podataka
- Nalaze se u prostoru imena System.Collections
- Mogu da se iskoriste samo za kretanje kroz postojeće podatke, a ne i za dodavanje novih podataka u kolekciju
- Razlika između IEnumerator i IEnumerable:

  Klasa koja implementira IEnumerator može da nabraja, obilazi podatke

  Klasa koja implementira IEnumerable ima metodu GetEnumerator, ta metoda vraća IEnumerator koji može da nabroji, obiđe podatke iz te klase

IEnumerator

```
public interface IEnumerator
{
    // Vraća trenutni element do koga smo došli prilikom obilaska kolekcije.
    object Current { get; }

    // Pomeranje na sledeći element u kolekciji. Vraća true ako je pomeranje
    // uspelo, vraća false ako nema više elemenata u kolekciji.
    bool MoveNext();

    // Pomera enumerator na poziciju pre početka kolekcije.
    void Reset();
}
```

IEnumerable

- public interface IEnumerable {

  // Metoda kreira i vraća enumerator koji obilazi elemente kolekcije.

  IEnumerator GetEnumerator();

  }
- Klasa koja implementira IEnumerator se obilazi korišćenjem metoda MoveNext() i Reset()
- Klasa koja implementira IEnumerable se obilazi korišćenjem foreach petlje:

  IEnumerable enumerable = new VektorEnumerable(4);

  …

  foreach (object o in enumerable)

  Console.WriteLine(o);
- Primer: projekat EnumeratorPrimer

Generički interfejsi IEnumerator<T> i IEnumerable<T>

- Interfejsi IEnumerator i IEnumerable rade sa kolekcijama čiji su elementi tipa Object
- Sa takvim kolekcijama javlja se potreba da se radi ispitivanje stvarnog tipa podataka i kasnije konverzija tipa
- Zbog toga imamo dodatan utrošak vremena i dodatnu mogućnost za izuzetak prilikom konverzije tipa
- Rešenje je korišćenje generičkih interfejsa IEnumerator<T> i IEnumerable<T>
- Ovi interfejsi se nalaze u prostoru imena System.Collections.Generic

IEnumerator<T>

- public interface IEnumerator<out T> : IDisposable, IEnumerator {

  // Vraća trenutni element do koga smo došli prilikom obilaska kolekcije.

  T Current { get; }

  }
- Generički interfejs IEnumerator<T> nasleđuje interfejse IEnumerator i IDisposable pa tako sadrži sve njihove metode i svojstva property-je i dodaje svoje svojstvo Current
- Klasa koja implementira IEnumerator<T> zato ima 2 svojstva sa nazivom Current – jedan tipa Object i drugi tipa T
- Dispose metoda koju nameće interfejs IDisposable može da se iskoristi za „čišćenje“ resursa jer se automatski zove posle završetka obilaska kolekcije foreach petljom (npr. za zatvaranje toka podataka kod enumeratora nad fajlovima). Ako nema spoljnih resursa za „čišćenje“ u enumeratoru onda Dispose metoda ostaje prazna (ovo je najčešći slučaj).

IEnumerable<T>

```
public interface IEnumerable<out T> : IEnumerable  {
    // Metoda kreira i vraća generički enumerator koji obilazi elemente
    //kolekcije.
    IEnumerator<T> GetEnumerator();
}
```

- IEnumerable<T> Nasleđuje „negenerički“ interfejs IEnumerable i dodaje svoju metodu GetEnumerator()
- Klasa koja implementira interfejs IEnumerable<T> zato ima dve GetEnumerator metode – jedna vraća IEnumerator<T>, a druga vraća IEnumerator
- Primer: projekat EnumeratorGenericPrimer

Ključna reč *yield*

- Služi za jednostavniju implementaciju IEnumerable interfejsa, direktno bez prethodne implementacije IEnumerator interfejsa
- Koristi se u kombinaciji sa return pa tako dobijamo yield return
- Smisao naredbe je da yield return ne znači konačan izlazak iz metode u pozivajuću metodu već označava „privremeni“ izlazak i istovremeno predstavlja tačku odakle počinje izvršenje metode prilkom narednog poziva
- Iteracija u pozivajućoj metodi se završava onda kada se izvrši poslednja yield return naredba u pozvanoj metodi ili se u pozvanoj metodi pozove yield break za kraj iteratora
- Metode koje imaju yield return naredbu moraju da imaju jedan od sledeća 4 povratna tipa: IEnumerable, IEnumerable<T>, IEnumerator ili IEnumerator<T>
- Primer: projekat PrimerYield i klase VektorEnumerableYield i VektorEnumerableYield<T> u prethodna dva projekta

Delegati (*delegates*)

- U engleskom jeziku *delegate* označava prosleđivanje nekog posla drugoj osobi (npr. direktor firme prosleđuje svoje poslove svojim zamenicima)
- U C#u delegat omogućava izbor metode koja će odraditi neki posao i to izbor u fazi izvršenja programa (*runtime* faza)
- U C/C++u postoji koncept pokazivača na funkcije
- Može se reći da su delegati u stvari objektno predstavljeni pokazivači na funkcije
- Način korišćenja:
  - 1. deklaracija delegata
  - 2. dodela jedne ili više metoda delegatu
  - 3. poziv izvršenja delegata

Delegati (*delegates*)

- Deklaracija delegata:
  - delegate <povratni tip> <naziv delegata> <lista parametara>
  - Primer: delegate int AritmetickaOperacija(int n);
  - Za ovakvu deklaraciju kompajler interno generiše klasu koja se zove npr. AritmetickaOperacija i izvedena je iz osnovne klase za predstavljanje delegata System.Delegate
- Primer: projekat Delegati

Delegati (*delegates*)

- Kreiranje instance delegata:
  - Primer: AritmetickaOperacija operacija1 = new AritmetickaOperacija(Dodaj);
  - Metoda Dodaj mora po povratnom tipu i listi argumenata da se slaže sa delegatom AritmetickaOperacija
  - Drugi način bez eksplicitnog poziva konstruktora (isti je efekat kao i u prethodnom slučaju, kompajler interno generiše poziv konstruktora):

    AritmetickaOperacija operacija2 = Pomnozi;
  - Treći način sa anonimnom metodom – interno kompajler generiše metodu bez imena kojoj mi ne možemo drugačije da pristupimo osim preko delegata:
  - Primer: AritmetickaOperacija operacija3 = delegate(int x) { return broj /= x; };
- Poziv izvršenja delegata:
  - Kreirani objekat tipa delegata se poziva metodom Invoke: operacija1.Invoke(25);
  - Potpuno ekvivalentan je i skraćeni način pisanja: operacija1(25);

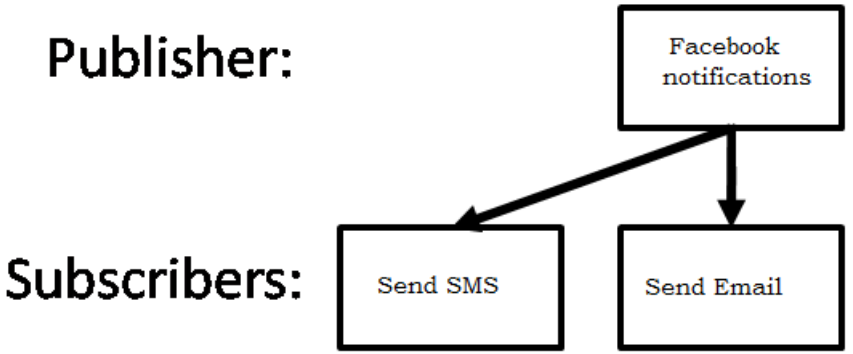
Slaganje delegata - *multicasting*

- Jednim pozivom može da se izvrši i više od jednog delegata
- Više delegata se međusobno povezuje operatorom **+** a zatim se zove izvršenje rezultujućeg delegata
- Operatori koji se povezuju operatorom + moraju biti istog tipa, a tog tipa će biti i dobijeni rezultat
- Iz rezultujućeg delegata bilo koji od njegovih „sastavnih delova“ može da se ukloni korišćenjem operatora –

- Primer: projekat SlaganjeDelegata
- Primer: projekat KoriscenjeDelegata

Zašto su nam potrebni delegati?

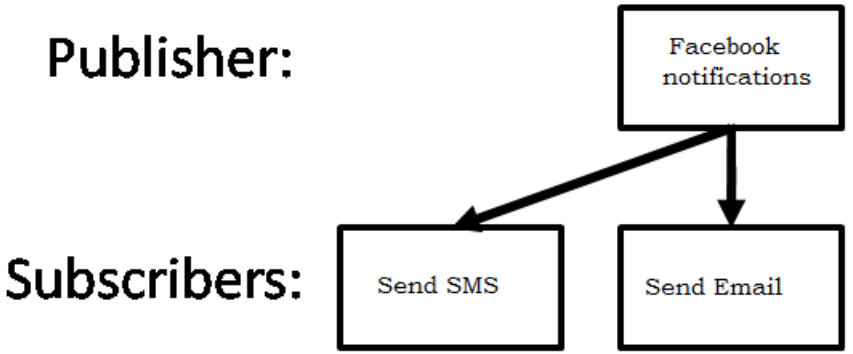
- Model publisher-subscriber



- Jedna klasa objavljuje događaj (*publisher*, generator događaja, objavljivač)
- Više drugih klasa može da osluškuje taj događaj i da izvrši neku akciju kad se događaj desi (klasa *subscriber*, potrošač događaja, pretplatnik)
- Primer: Klasa koja šalje obaveštenja o nekom FB događaju je *publisher*. Na događaje te klase može biti pretplaćeno više drugih klasa – *subscribers*. U primeru su to klase SendSMS za prosleđivanje notifikacije SMS-om i SendEmail za prosleđivanje notifikacije e-mail-om.

Zašto su nam potrebni delegati?

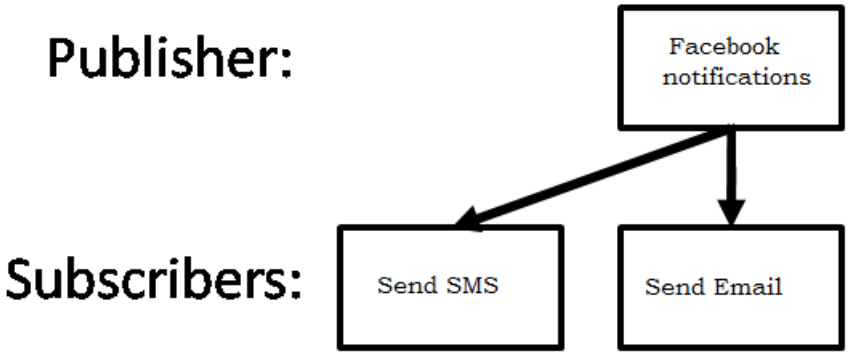
- Model publisher-subscriber



- Koncept delegata omogućuje da Publisher „ne zna“ detalje o implementaciji Subscriber-a (npr. Publisher može da bude gotova klasa iz .NET Framework biblioteke, a mi sami možemo da implementiramo proizvoljno mnogo Subscriber metoda u jednoj ili više klasa)
- Primer: projekat PublisherSubscriber1
- Problem: Publisher dodaje subscriber-a, više liči na model Master-Slave nego Publisher-Subscriber

Zašto su nam potrebni delegati?

- Model publisher-subscriber



- Primer: projekat PublisherSubscriber2
- Rešava prethodni problem – subscriber sam odlučuje da li će se pretplatiti
- Problem: bilo koji objekat može da pozove ili promeni delegat iz publishera
- Rešenje prethodnih problema – događaji (*events*)
- Primer: projekat PublisherSubscriber3

Događaji (*event*)

- Događaji predstavljaju posebnu vrstu delegata
- Služe nam za implementaciju modela *publisher* (generator događaja, objavljivač) – *subscriber* (potrošač događaja, pretplatnik)
- Generator događaja (*publisher*):
  - Objekat koji sadrži definicije događaja i delegata
  - U ovom objektu se zadaje veza događaj – delegat
  - Ovaj objekat pokreće događaj pa se potrošači događaja obaveštavaju o tome
- Potrošač događaja (*subscriber*):
  - Objekat koji prihvata događaj i sadrži metodu za obradu događaja (*event handler*)
  - Delegat u objektu generatoru događaja poziva na izvršenje metodu za obradu događaja u potrošaču događaja

Događaji (*event*)

- Sintaksa
- Najpre se definiše delegat:
  - public delegate void ObradaPromeneBroja(int broj);
- Zatim se definiše dogadjaj za taj tip delegata:
  - public event ObradaPromeneBroja PromenaBroja;
- Kasnije se događaju (*event*-u) dodeljuju ili oduzimaju metode koje ga obrađuju **isključivo korišćenjem operatora += i -=**

- Primer: projekat Dogadjaji

Struktura DateTime

- Nalazi se u prostoru imena System
- Služi za čuvanje informacija o datumu i vremenu u toku tog datuma
- Vremenska informacija je predstavljena „tikovima“ – jedan „tik“ odgovara vremenskom intervalu od 100ns (10^{-7} s)
- Cela stuktura zauzima 64 bita, a od toga se 62 bita koriste za čuvanje broja „tikova“
- Moguće je predstaviti vrednosti od 1.1.0001. nove ere u 0:00:00 do 31.12.9999. 23:59:59.9999999 tj. vrednost od nula „tikova“ odgovara trenutku 1.1.0001. u 0:00:00
- Detaljnije objašnjenje metoda i property-ja ove struktrure dato je u projektu DateTimeStruktura

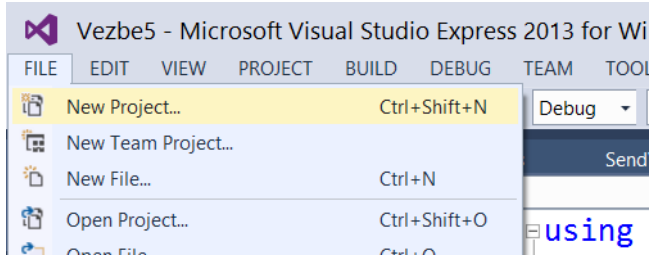
Struktura TimeSpan

- Nalazi se u prostoru imena System
- Dobija se primenom operatora minus – na dve instance strukture DateTime
- DateTime predstvlja jedan vremenski trenutak, a TimeSpan predstvlja vremenski interval

Windows forme

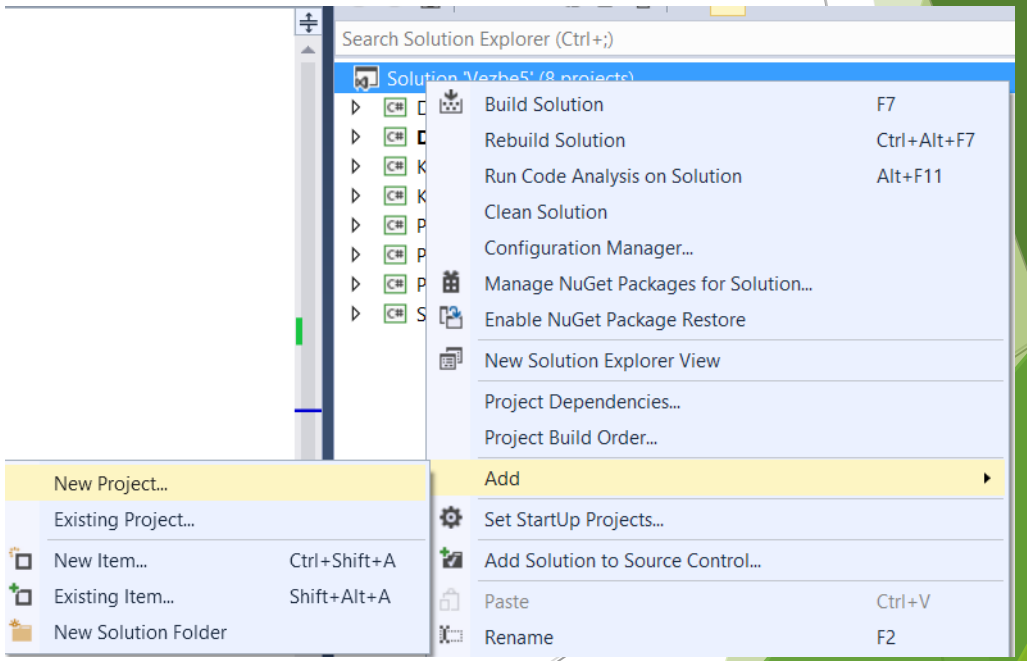
Kreiranje Windows aplikacije

- Windows aplikacija se kreira kao poseban tip aplikacije u Visual Studiju
- Potrebno je najpre kreirati novi projekat na jedan od dva moguća načina
- 1. Menu->File->New Project…



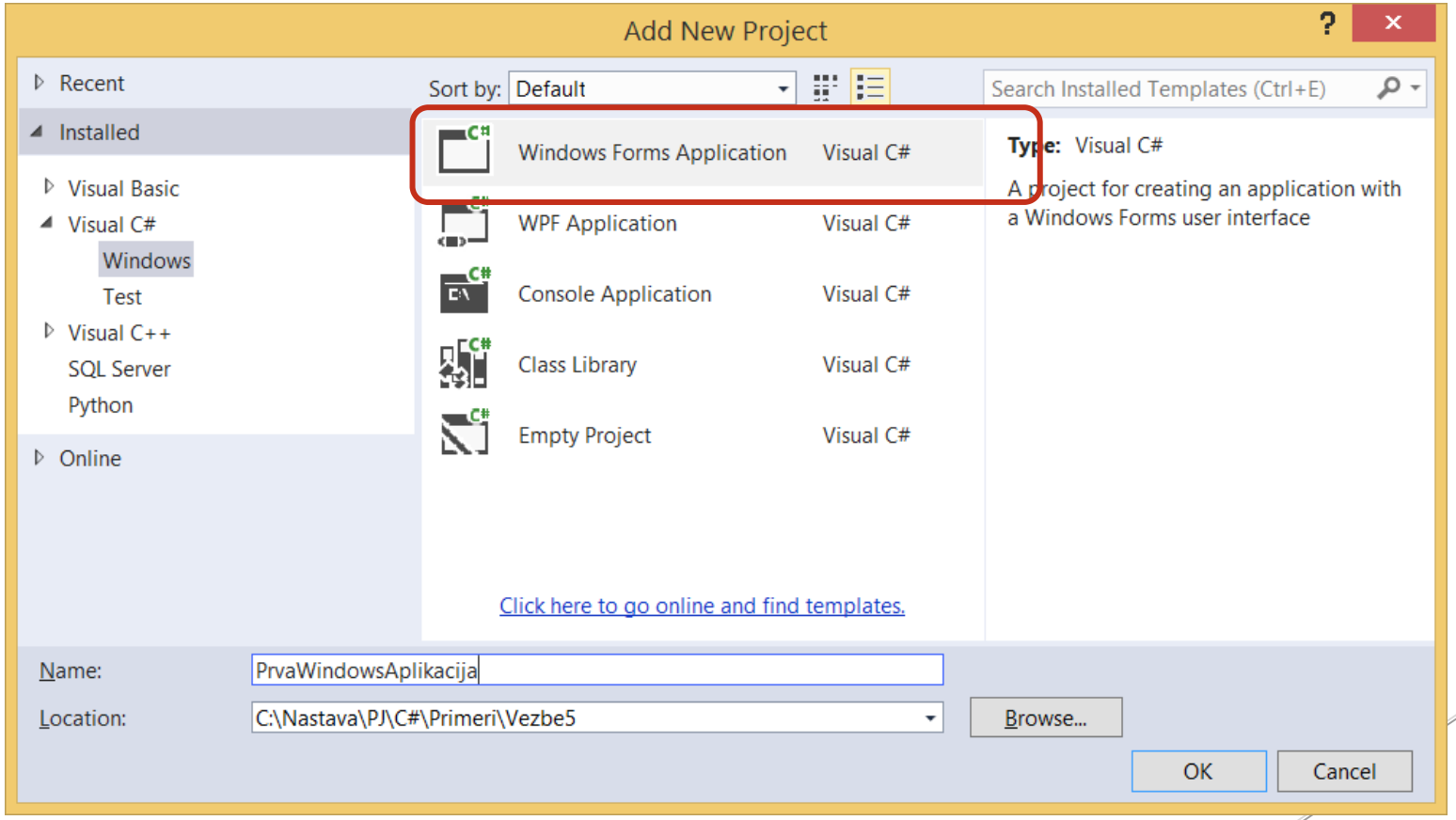
- 2. Desni klik na postojeći

solution, pa zatim opcija Add->New Project…



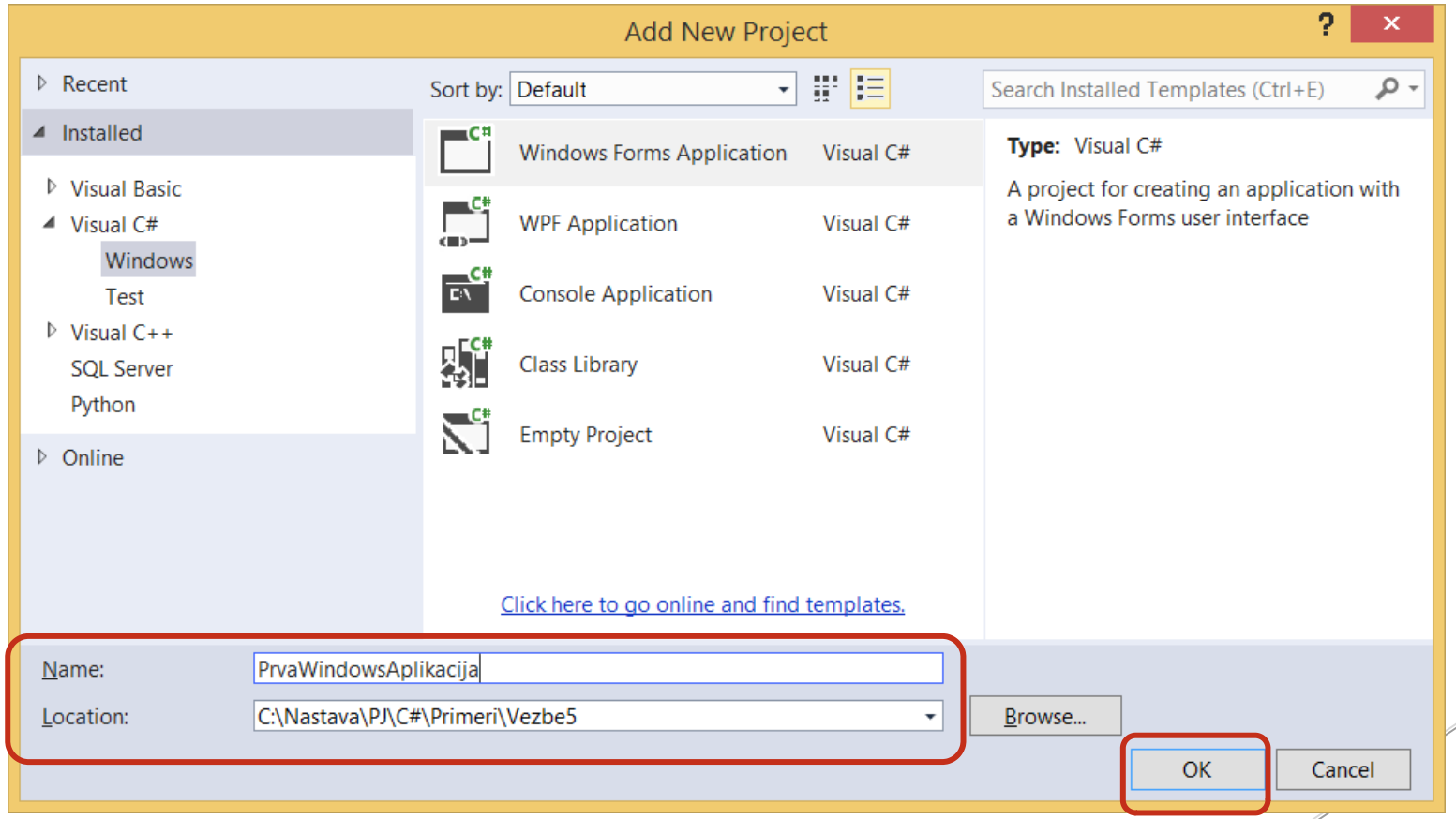
Kreiranje Windows aplikacije

- U dijalogu za kreiranje novog projekta traba izabrati „Windows Forms Application“



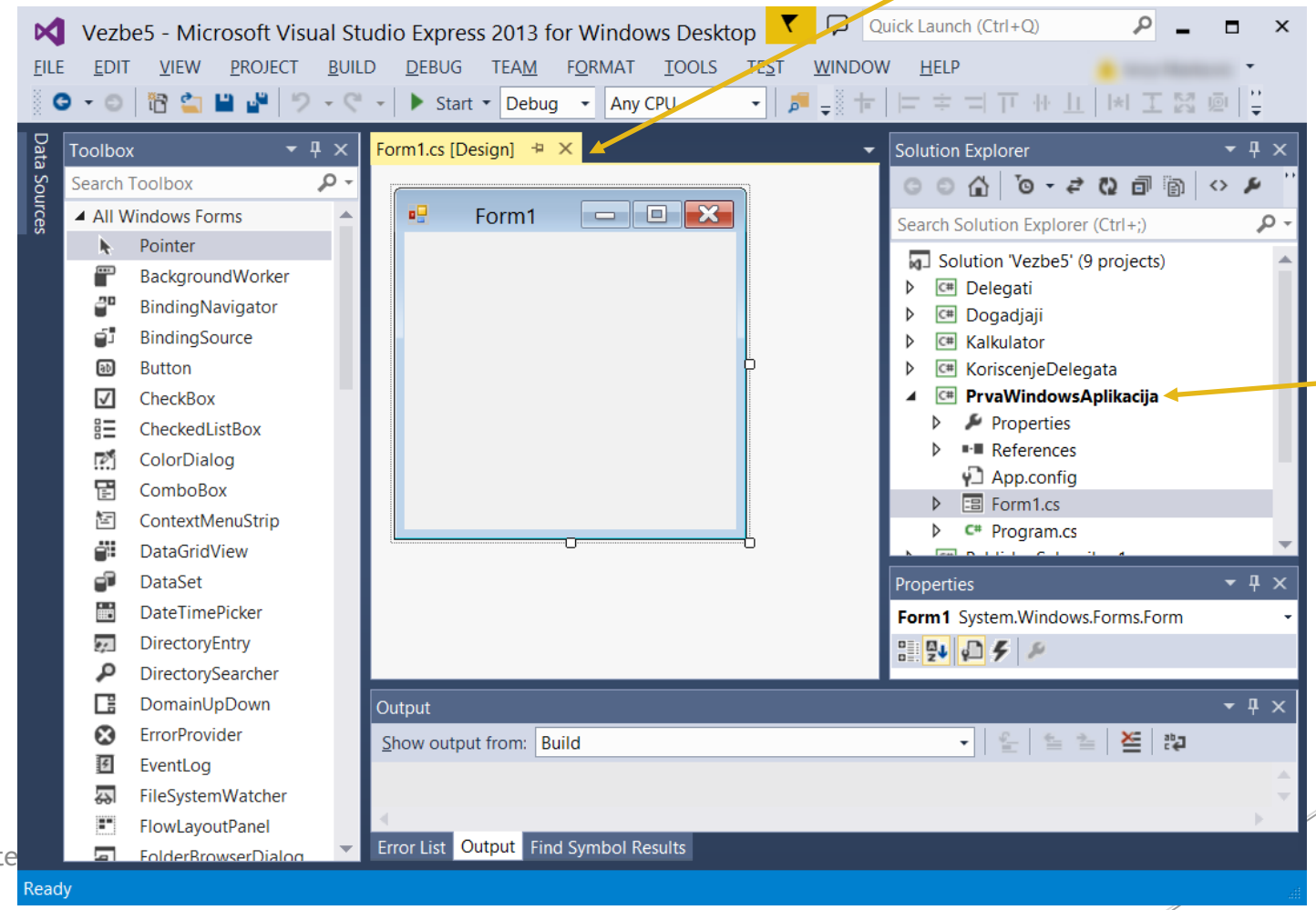
Kreiranje Windows aplikacije

- Zatim podesiti naziv projekta i lokaciju pa kliknuti na dugme „OK“



Kreiranje Windows aplikacije

- Dobija se pogled kao na slici

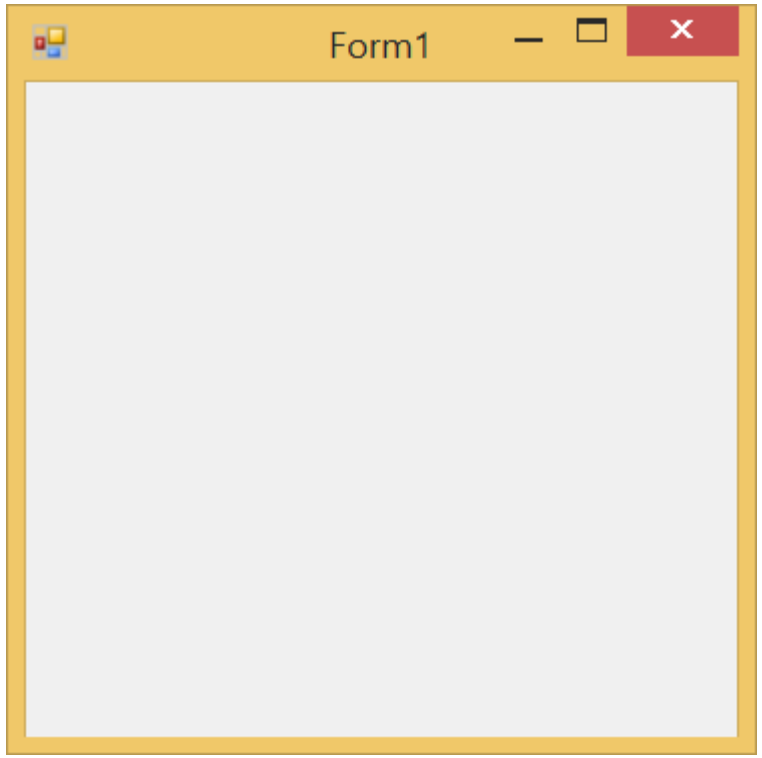


Centralni deo prikazuje dizajn tj. izgled prozora kreirane aplikacije

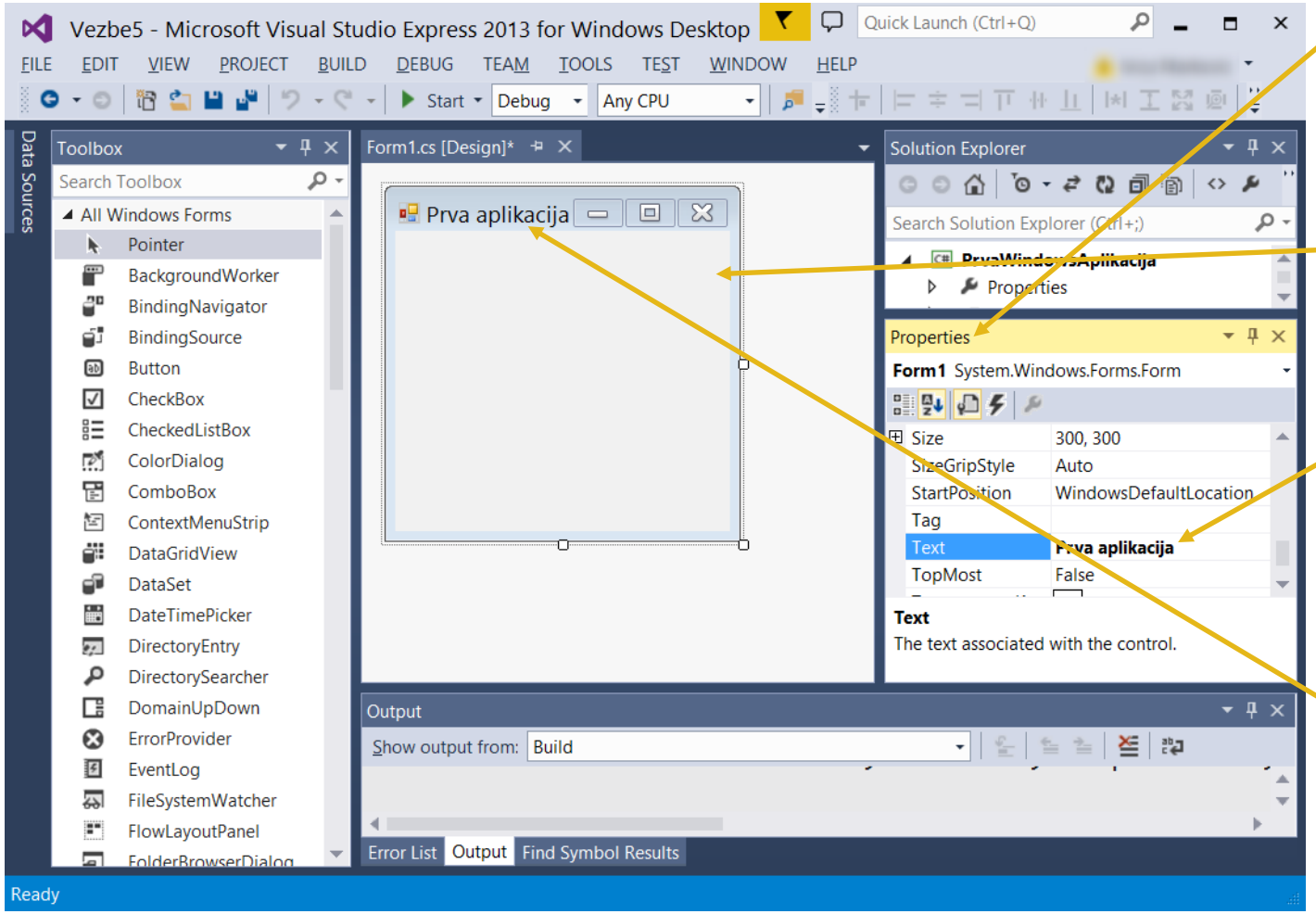
Solution Explorer sadrži kreirani projekat

Kreiranje Windows aplikacije

- Po pokretanju aplikacije prikazuje se prozor kao na slici



Kreiranje Windows aplikacije



Podešavanja forme (prozora) i kontrola unutar prozora radi se u Properties pogledu

1. Selektujemo formu u Design pogledu

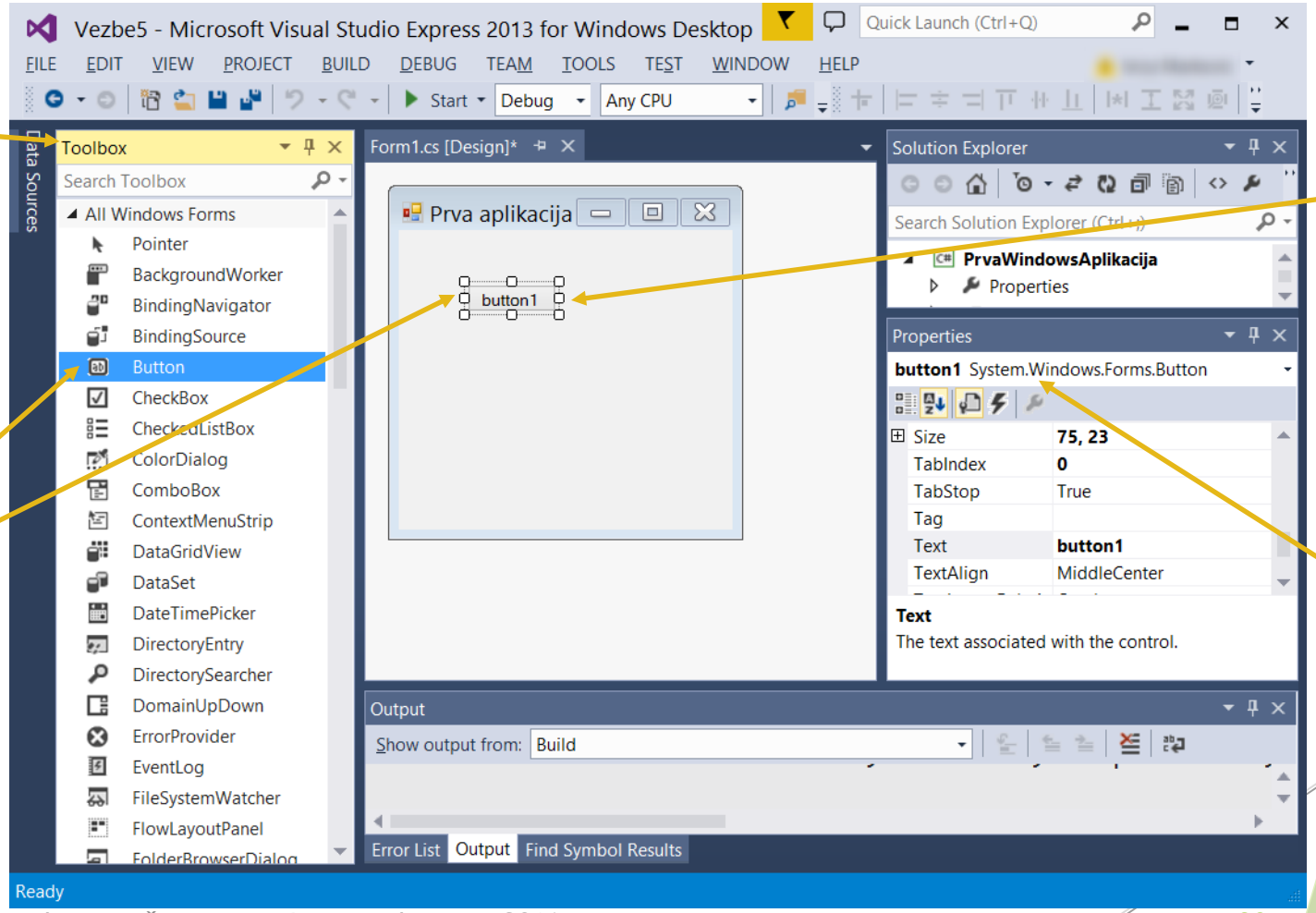
2. Promenimo property „Text“

3. Taj tekst predstavlja natpis na prozoru aplikacije

Kreiranje Windows aplikacije

1. Toolbox pogled sadrži kontrole koje možemo dodati našoj formi

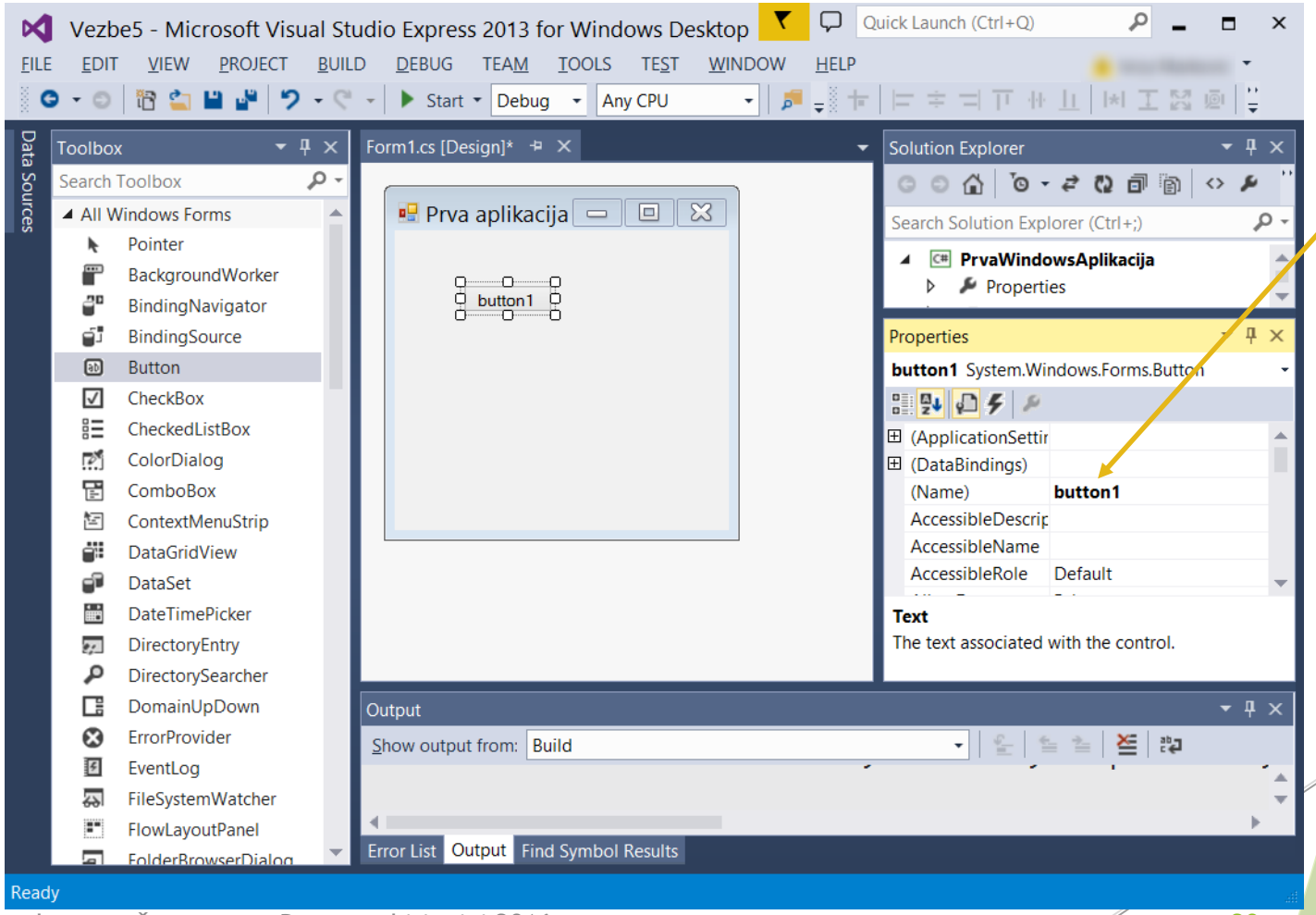
2. Biramo kontrolu tipa dugme (Button) i mišem je prevlačimo na našu formu



3. Kontrola dugme je selektovana na formi

4. Properties pogled sada prikazuje podatke o toj kontroli

Kreiranje Windows aplikacije

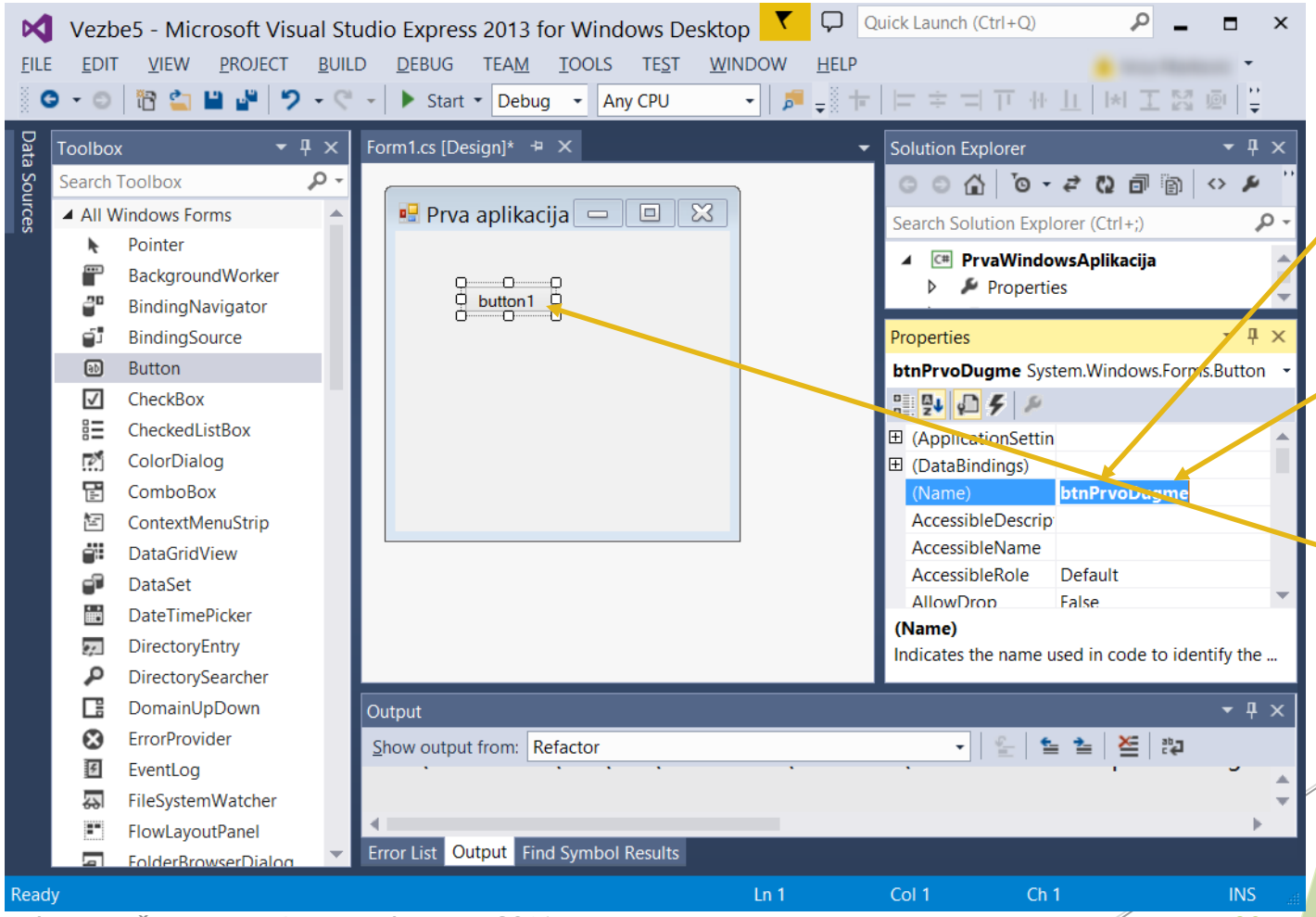


1. Property (Name) prikazuje automatski generisano ime dugmeta “button1”

2. To je istovremeno i naziv promenljive koja predstavlja ovu instancu klase Button

3. Ovakvo ime ODMAH treba promeniti u neko smisleno ime zbog preglednosti koda

Kreiranje Windows aplikacije

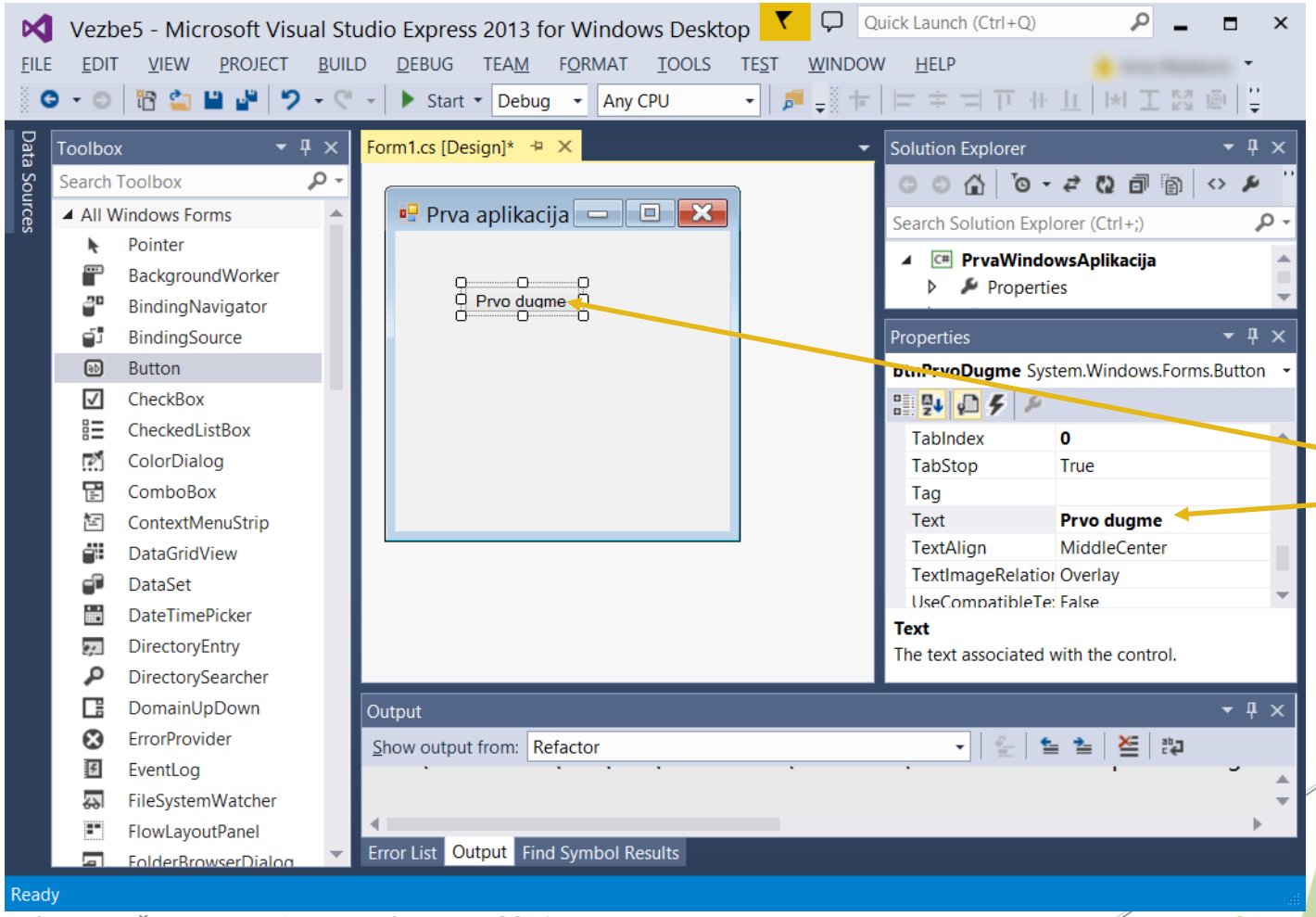


4. Dobra praksa je da ime instance kontrole sadrži troslovnu skraćenicu tipa kontrole – u ovom slučaju btn od Button

5. Biramo ime npr. btnPrvoDugme

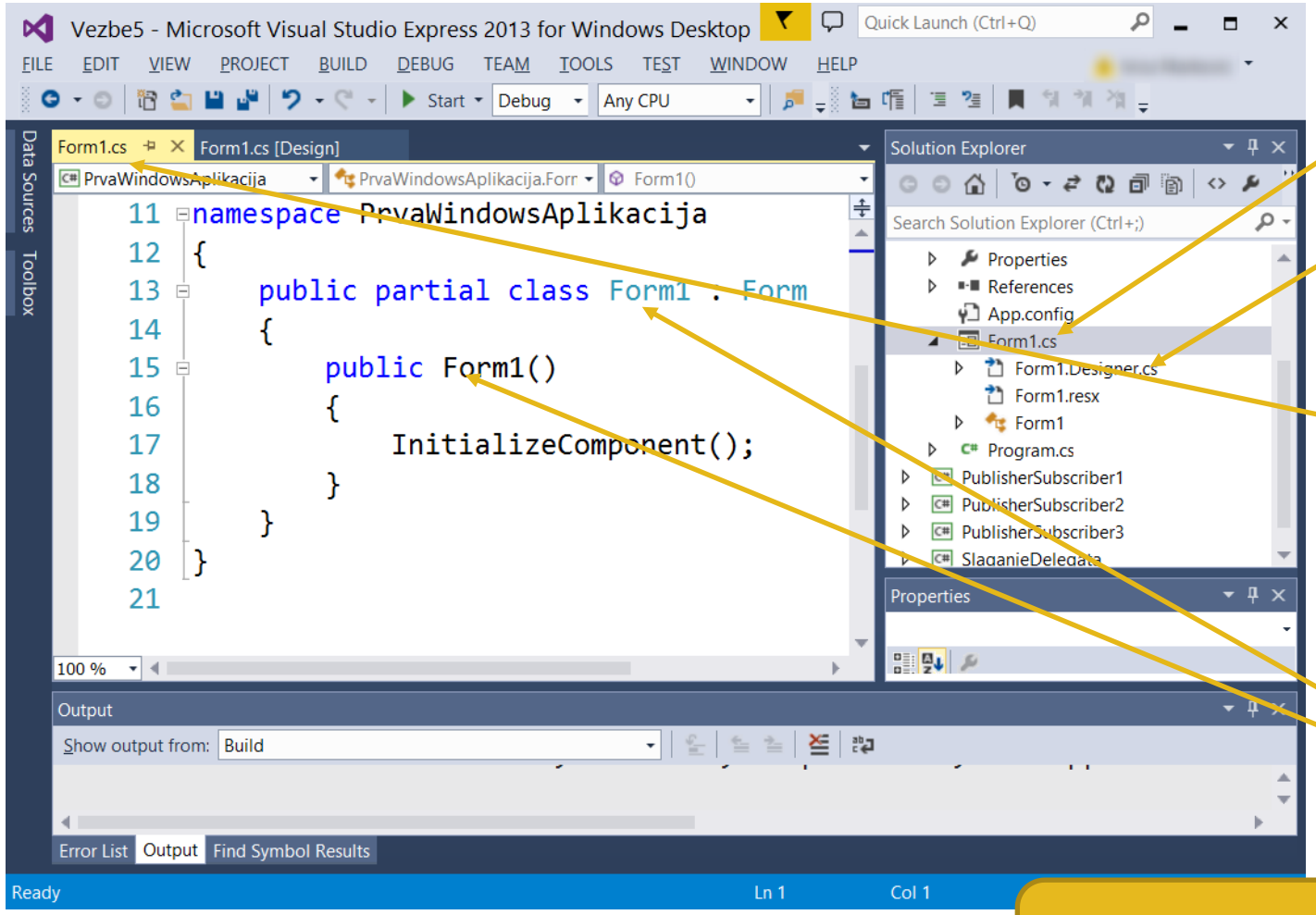
6. (Name) property ne utiče na natpis na dugmetu

Kreiranje Windows aplikacije



7. Natpis na kontroli određen je property-jem koji se zove Text

Kreiranje Windows aplikacije



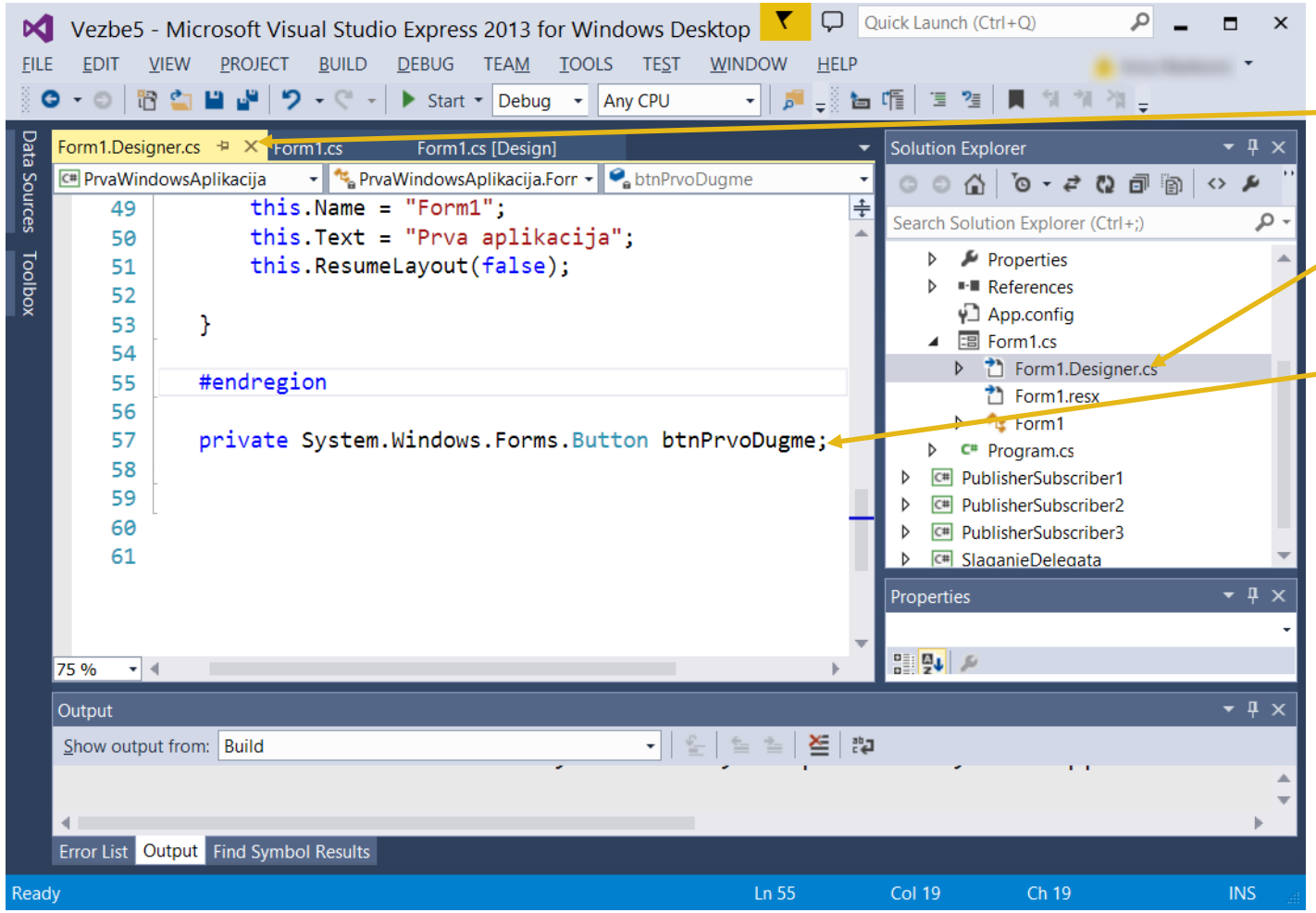
1. U Solution Explorer pogledu vidimo 2 fajla sa C# programskim kodom: Form1.cs i Form1.Designer.cs

2. Desni klik u Solution Explorer pogledu na Form1.cs, opcija „View code“ otvara nam sadržaj fajla Form1.cs u VS editoru

3. Tu se nalazi deo parcijalne klase Form1 izvedene iz klase System.Windows.Forms.Form koja u početku u ovom fajlu ima samo konstruktor

4. Form1.cs je mesto gde možemo da dodajemo svoj programski kod klasi Form1

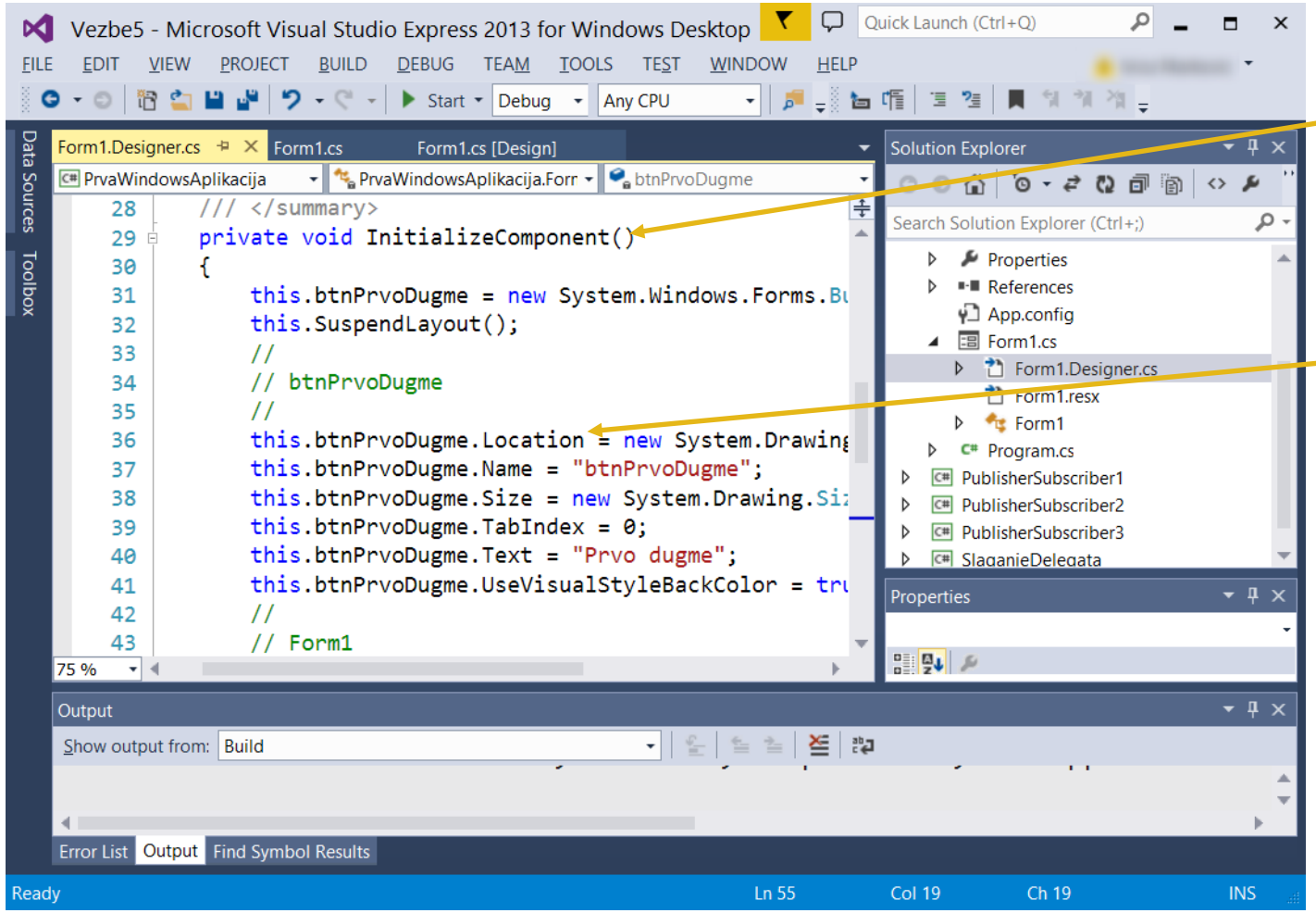
Kreiranje Windows aplikacije



1. Dvoklikom u Solution Explorer pogledu na Form1.Designer.cs otvaramo sadržaj fajla Form1.Designer.cs u VS editoru

2. Ovde se nalaze atributi klase Form1 koji predstavljaju kontrole na njoj, a dodali smo ih prevlačenjem mišem iz Toolbox pogleda

Kreiranje Windows aplikacije



3. Primećujemo i implementaciju privatne metode InitializeComponent() koju poziva konstruktor Form1() u fajlu Form1.cs

4. U medodi InitializeComponent() VS generiše programski kod na osnovu naših podešavanja iz dizajner pogleda (npr. Location i Size za dugme) i iz Properties pogleda (npr. Name i Text)

5. Programski kod u fajlu Form1.Designer.cs NE EDITUJEMO RUČNO jer bi pri narednom generisanju koda naše promene bile izgubljene

Kreiranje Windows aplikacije

- Za obradu akcija od strane korisnika (npr. klik na dugme, klik mišem na neku kontrolu/formu, izbor neke stavke u meniju itd.) koriste se događaji (event) i delegati (delegate)
- Klasa koja predstvlja kontrolu (npr. za dugme je klasa Button) ima javne događaje kojima možemo da dodelimo delegate preko kojih pozivamo naše metode za obradu događaja.
-
```
public class Button : ButtonBase, IButtonControl  {
    …
    // pored ostalih članova klasa Button sadrži i event za klik (u stvari taj event je
    // u klasi Control iz koje je izvedena klasa Button, ali za ovaj primer to nije važno)
    public event EventHandler Click;
    …
}
```
- Primer: projekti PrvaWindowsAplikacija i Kalkulator

Kreiranje Windows aplikacije

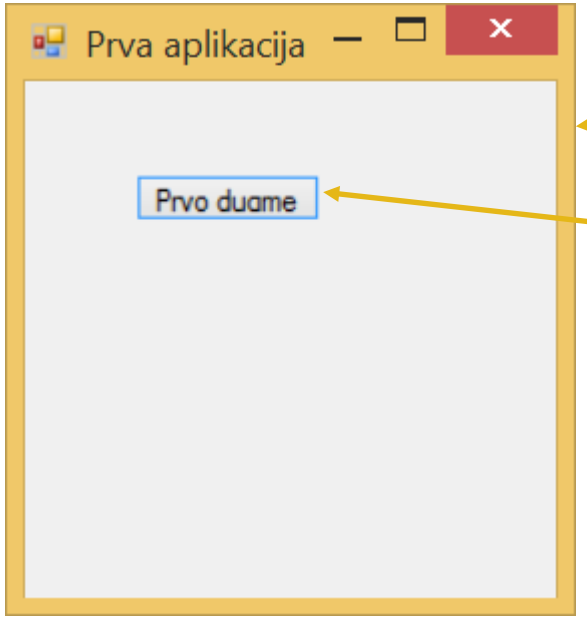
- U prostoru imena System postoji delegat EventHandler definisan ovako

  public delegate void EventHandler(object sender, EventArgs e);
- Ovo je delegat za korisničku metodu koja će izvršiti obradu događaja, metoda mora da se slaže u povratnom tipu (void) i tipovima argumenata sa ovim delegatom
- Prvi argument object sender je sama instanca kontrole koja je generisala događaj
- Drugi argument EventArgs e služi za prenošenje dodatnih informacija o događaju (argumenti događaja)
- Zavisno od vrste događaja drugi argument može da bude i instanca neke klase izvedene iz klase EventArgs i da nosi neke specifične informacije (npr. koordinate gde se desio klik mišem, oznaku tastera koji je pritisnut na tastaturi itd.)
- Slede važni delovi koda u našoj klasi Form1 za obradu klika na dugme (ne ulazimo u detalje u kom delu parcijalne klase je šta implementirano)

Kreiranje Windows aplikacije

```
public class Form1 : Form  {
  private Button btnPrvoDugme;  // ovo generiše VS kada prevučemo dugme na formu
  …
  this.btnPrvoDugme.Click += new System.EventHandler(this.btnPrvoDugme_Click);
  // ovu liniju generiše VS kada u Properties pogledu dodamo event handler za Click event
  …
  // i sledeću metodu generiše VS kada u Properties pogledu dodamo event handler za Click event
  private void btnPrvoDugme_Click(object sender, EventArgs e){
    // generisana metoda je prazna pa unutar nje možemo da pišemo sopstveni kod
    MessageBox.Show("Klik na dugme!");  // dodali smo prikazivanje dijaloga sa zadatom porukom
  }  }
```

Kreiranje Windows aplikacije

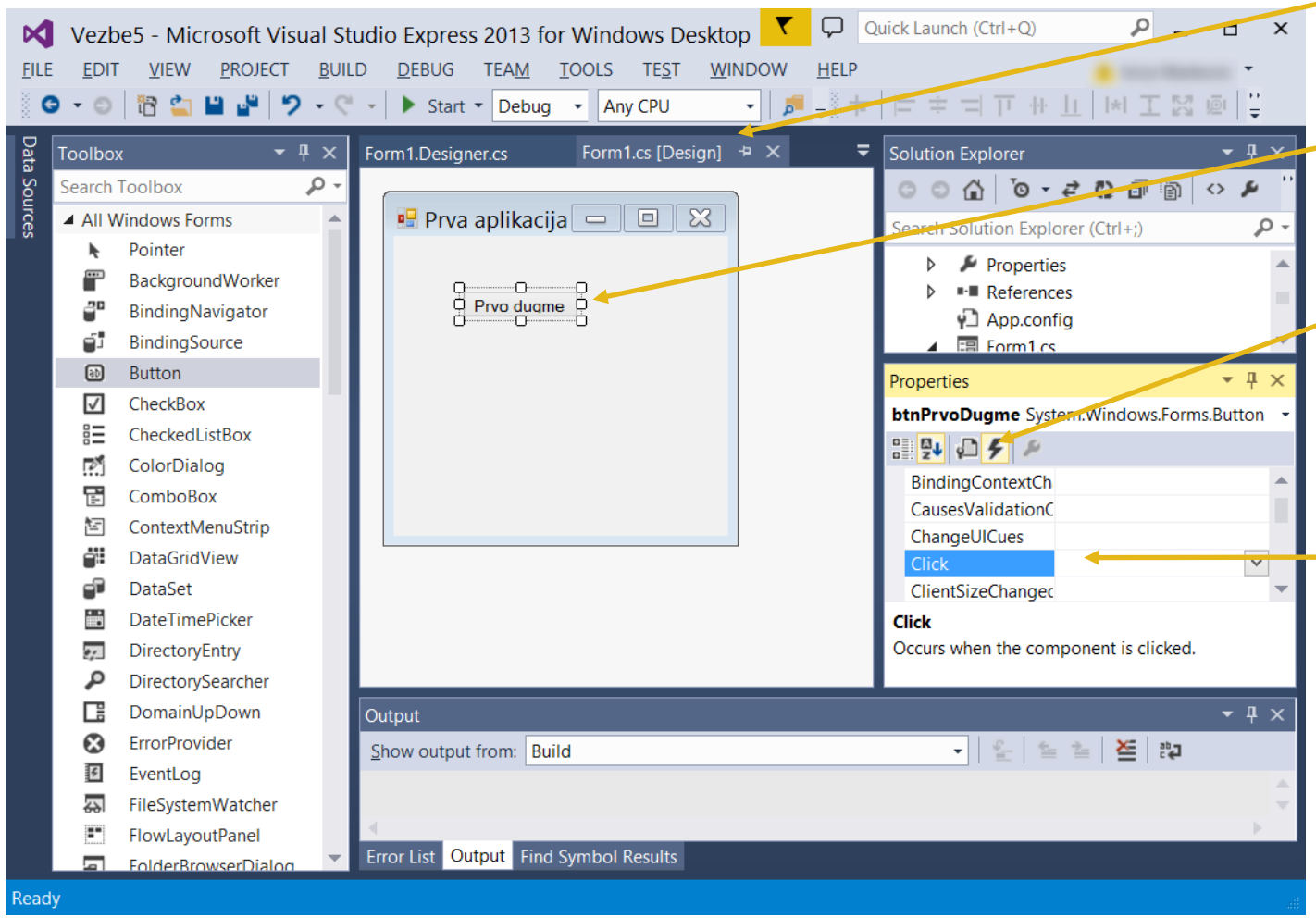


1. Ovaj prozor je rezultat pokretanja našeg projekta

2. Klik na dugme nema nikakvog efekta

3. Nedostaje nam metoda koja će služiti kao *event handler* događaju Click za promenljivu btnPrvoDugme

Kreiranje Windows aplikacije



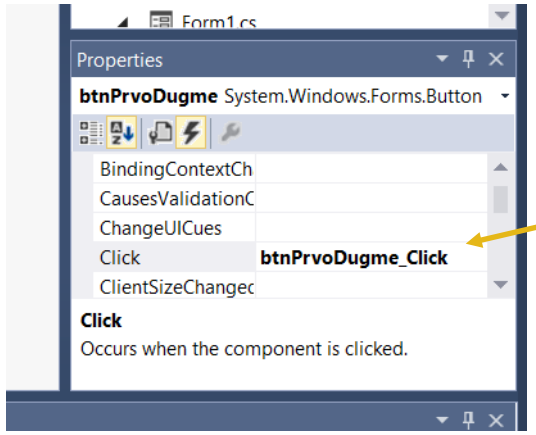
4. Prelazimo na tab Form1.cs[Design] – dizajner pogled

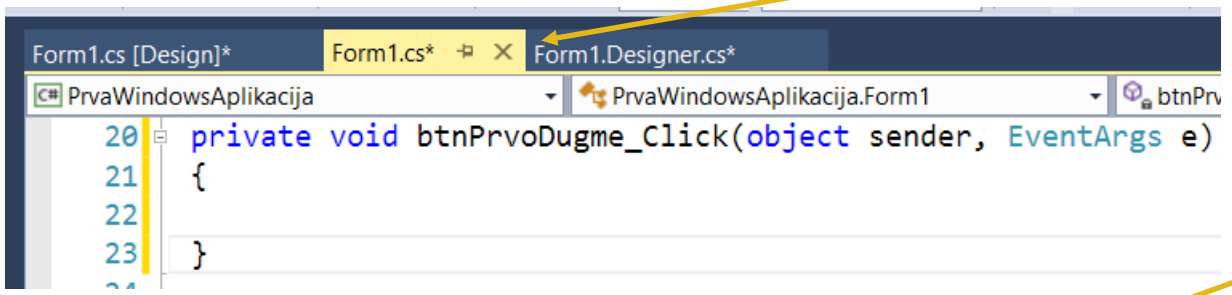
5. Selektujemo dugme

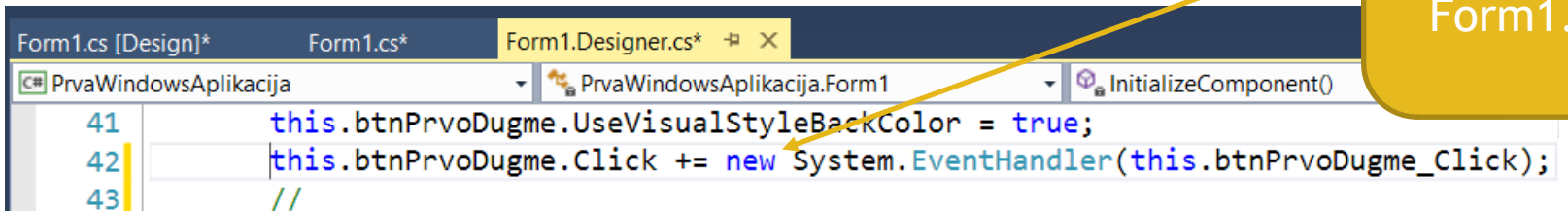
6. U properties pogledu biramo tab sa događajima (*events*) klikom na simbol „munja“

7. Uradimo dvoklik na događaj Click u listi

Kreiranje Windows aplikacije







8. VS generiše metodu za obradu događaja – event handler i dodeljuje joj naziv po šablonu <ime_promenljive>_<ime_događaja>

9. Generisana prazna metoda se nalazi u fajlu Form1.cs i tu možemo da zadamo svoj kod za obradu događaja

10. Povezivanje generisane metode na *event* Click promenljive btnPrvoDugme automatski je odrađeno u fajlu Form1.Designer.cs i to NE TREBA RUČNO DA MENJAMO

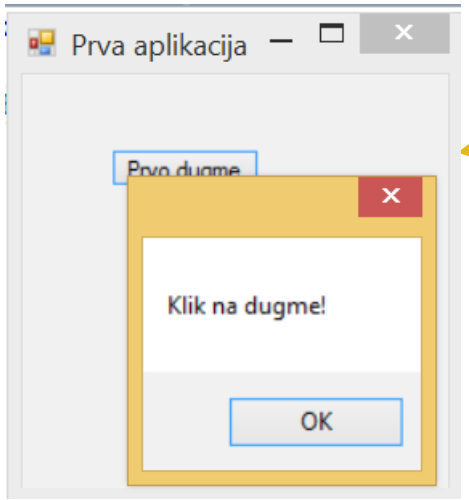
Kreiranje Windows aplikacije

Form1.cs [Design] | Form1.cs | Form1.Designer.cs

PrvaWindowsAplikacija | PrvaWindowsAplikacija.Form1 | btnPrvoD

```
20  private void btnPrvoDugme_Click(object sender, EventArgs e)
21  {
22      MessageBox.Show("Klik na dugme!");
23  }
24
```

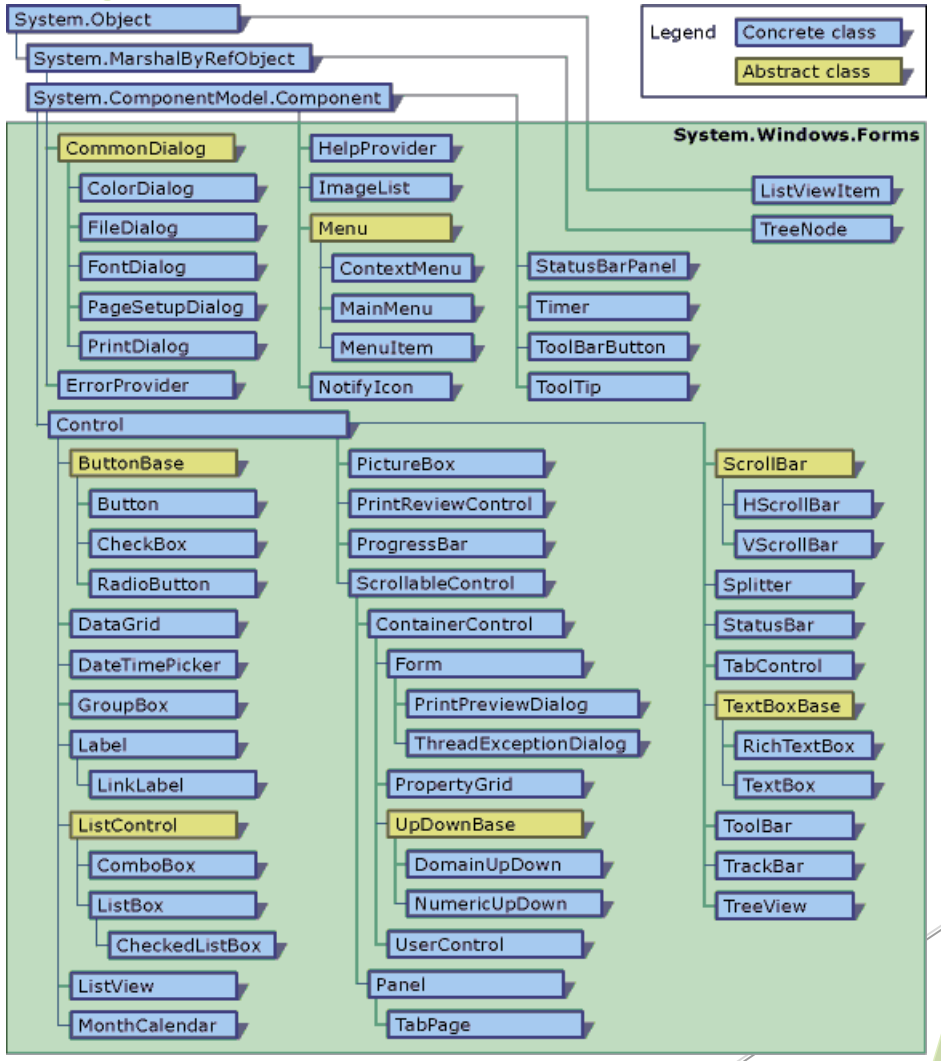
11. U metodi za obradu događaja dodajemo neki programski kod npr. prikazivanje poruke o kliku na dugme



12. Sada naša aplikacija posle klika na dugme izbacuje poruku

System.Windows.Forms namespace

- Sadrži klase za kreiranje Windows aplikacija
- Najvažnije klase su Control i Form
- Form je klasa koja predstavlja prozor aplikacije
- I ona je izvedena iz klase Control



Klasa Control

- Osnovna klasa koja definiše elemente za vizualno predstavljanje kontrola, kao i neke zajedničke elemente njihovog ponašanja
- Property-ji za definisanje hijerarhije među kontrolama (kontrola koja sadrži drugu kontrolu se smatra roditeljskom kontrolom za nju, sadržana kontrola se smatra potomkom kontrole u okviru koje se nalazi)
  - Control.ControlCollection Controls { get; }
  - Control Parent { get; set; }
  - bool HasChildren { get; }
- Property-ji za definisanje vidljivosti kontrole
  - bool Visible { get; set; } – zadaje vidljivost kontrole i njenih *child* kontrola
  - bool Enabled { get; set; } – true kontrola prihvata akcije korisnika, false kontrola je i dalje vidljiva ali ne prihvata akcije korisnika

Klasa Control

- Property-ji za definisanje fokusa kontrole
  - bool Focused { get; } – da li kontrola ima fokus za unos od strane korisnika
  - bool TabStop { get; set; } – da li korisnik tasterom TAB može da se pozicionira na kontrolu tj. da joj da fokus
  - int TabIndex { get; set; } – određuje redosled kojim se kontrole fokusiraju pritiskom na taster TAB
- Property-ji za definisanje veličine kontrole
  - int Width { get; set; } – širina kontrole u pikselima
  - int Height { get; set; } – visina kontrole u pikselima
  - Size Size { get; set; } – property tipa strukture System.Drawing.Size koja takođe sadrži property-je Width i Height, objedinjuje prethodna 2 property-ja

Klasa Control

- Property-ji za definisanje lokacije kontrole
  - Point Location { get; set; } – property tipa strukture System.Drawing.Point koja sadrži property-je int X i int Y koji predstavljaju koordinate gornjeg levog ugla naše kontrole u odnosu na njenu roditeljsku kontrolu
- Property-ji koji zadaju koordinate stranica pravougaonika koji ograničava kontrolu u odnosu na levu i gornju stranicu roditeljske kontrole
  - int Left { get; set; }
  - int Right { get; }
  - int Top { get; set; }
  - int Bottom { get; }

Klasa Control

- Property-ji za definisanje maksimalne i minimalne veličine kontrole
  - Size MaximumSize { get; set; }
  - Size MinimumSize { get; set; }

Klasa Control

- Property-ji za definisanje načina povezivanja kontrole sa roditeljskom kontrolom
  - AnchorStyles Anchor { get; set; } – property tipa enumeracije System.Windows.Forms.AnchorStyles čije vrednosti se mogu kombinovati bitskim „ili“ operatorom |
  - Vrednosti su None = 0, Top = 1, Bottom = 2, Left = 4, Right = 8
  - Određuju koja stranica kontrole je „usidrena“ za koju stranicu roditeljske kontrole na rastojanju zadatom property-jima Control.Top, Control.Bottom, Control.Left i Control.Right
  - Obratiti pažnju na razliku između property-ja klase Control (npr. Control.Top) i vrednosti AnchorStyles enumeracije (npr. AnchorStyles.Top)
  - Default vrednosti su Control.Anchor = AnchorStyles.Top | AnchorStyles.Left

Klasa Control

- Property-ji za definisanje načina povezivanja kontrole sa roditeljskom kontrolom
  - DockStyle Dock { get; set; } – property tipa strukture System.Windows.Forms.DockStyle
  - Vrednosti su None, Top, Bottom, Left, Right, Fill
  - Određuju koja stranica kontrole je “zalepljena” za koju stranicu roditeljske kontrole na rastojanju 0
  - Default vrednosti su Control.Dock = AnchorStyles.None

Klasa Control – važni događaji

- Neki od događaja iz klase Control i odgovarajući delegati koji ih obrađuju
  - Click => EventHandler: public event EventHandler Click;
  - MouseDown => MouseEventHandler: public event MouseEventHandler MouseDown;
  - MouseUp => MouseEventHandler: public event MouseEventHandler MouseUp;
  - MouseMove => MouseEventHandler: public event MouseEventHandler MouseMove;
  - MouseWheel => MouseEventHandler: public event MouseEventHandler MouseWheel;
  - KeyDown => KeyEventHandler: public event KeyEventHandler KeyDown;
  - KeyPress => KeyPressEventHandler: public event KeyPressEventHandler KeyPress;
  - KeyUp => KeyEventHandler: public event KeyEventHandler KeyUp;

Delegat EventHandler

- public delegate void EventHandler(object sender, EventArgs e);
- Najjednostavniji delegat za događaje kontrola
- Argumenti događaja se prenose pomoću klase EventArgs
- Klasa EventArgs ne nosi nikakve dodatne informacije od događaju i koristi se za jednostavne događaje kada ne treba da se prenese nikakva informacija (osim da se događaj desio)
- Klasa EventArgs je nadklasa svim ostalim klasama za prenos argumenata događaja

Delegat MouseEventHandler

- public delegate void MouseEventHandler(object sender, MouseEventArgs e);
- Događaji ovog tipa su MouseDown, MouseUp, MouseMove, MouseWheel
- Argumenti događaja se prenose pomoću klase MouseEventArgs koja je izvedena iz klase EventArgs

Delegat MouseEventHandler

- public delegate void MouseEventHandler(object sender, MouseEventArgs e);
- Klasa MouseEventArgs sadrži sledeće property-je:
  - public MouseButtons Button { get; } // vraća koje dugme je pritisnuto
  - public int Clicks { get; } // vraća koliko puta je dugme miša pritisnuto i otpušteno
  - public int Delta { get; } // vraća broj „zareza“ za koliko je okrenut točkić miša
  - public Point Location { get; } // vraća lokaciju kursora miša u trenutku izvršenja akcije
  - public int X { get; } // vraća X komponentu lokacije kursora miša u trenutku izvršenja akcije
  - public int Y { get; } // vraća Y komponentu lokacije kursora miša u trenutku izvršenja akcije

Događaji tastature

- Za rad sa tastaturom služe sledeći događaji koji se izvršavaju u ovom redosledu u kom su i navedeni:
  - public event KeyEventHandler KeyDown; // okida se kada je taster pritisnut (u donjem položaju)
  - public event KeyPressEventHandler KeyPress; // okida se kada je taster pritisnut pa otpušten (kada se vrati u gornji položaj)
  - public event KeyEventHandler KeyUp; // okida se kada je taster pritisnut pa otpušten (kada se vrati u gornji položaj) posle događaja KeyPress

Delegat KeyPressEventHandler

- public delegate void KeyPressEventHandler(object sender, KeyPressEventArgs e);
- Događaj ovog tipa je KeyPress
- Argumenti događaja se prenose pomoću klase KeyPressEventArgs koja je izvedena iz klase EventArgs
- Klasa KeyPressEventArgs sadrži sledeće property-je:
  - public bool Handled { get; set; } // Označava da li je kontrola već obradila taj događaj. Po default-u je false. Ako npr. u KeyPress event handleru u TextBox kontroli postavimo ovu vrednost na true pritisnuti karakter se neće prikazati u TextBox-u.
  - public char KeyChar { get; set; } // Karakter koji je otkucan na tastaturi.

Delegat KeyEventHandler

- public delegate void KeyEventHandler(object sender, KeyEventArgs e);
- Događaji ovog tipa su KeyDown i KeyUp
- Argumenti događaja se prenose pomoću klase KeyEventArgs koja je izvedena iz klase EventArgs
- Klasa KeyEventArgs sadrži sledeće property-je:
  - public virtual bool Alt { get; } // true ako je pritisnut taster Alt, false inače
  - public bool Control { get; } // true ako je pritisnut taster Ctrl, false inače
  - public virtual bool Shift { get; } // true ako je pritisnut taster Shift, false inače
  - public bool Handled { get; set; } // za izbegavenje podrazumevanog ponašanja kontrole prilikom pritiska tastera ovu vrednost treba postaviti na true
  - public bool SuppressKeyPress { get; set; } // za izbegavenje podrazumevanog ponašanja kontrole prilikom pritiska tastera ovu vrednost treba postaviti na true

Delegat KeyEventHandler

- Enumeracija Keys služi da označi koji taster je pritisnut.
- Vrednosti ove enumeracije od 1 do 254 služe za predstavljanje nekog od „običnih“ tastera (A, B, C, D, D1, D2,…, D0, E, F, F1, F2,…, H, Home, End, Left, Right, PrintScreen,…)
- Vrednosti ove enumeracije za predstavljanje tastera modifikatora su Shift = 65536, Control = 131072, Alt = 262144 (u binarnom zapisu modifikatori imaju samo jednu jedinicu i sve ostale nule pa tako mogu da se spajaju međusobno i/ili sa najviše jednim „običnim“ tasterom pomoću bitske OR operacije)

Delegat KeyEventHandler

- Klasa KeyEventArgs sadrži sledeće property-je:
  - public Keys Modifiers { get; } // vraća podatak o pritisnutim modifikatorima Alt, Ctrl i Shift (1, 2 ili 3 modifikatora istovremeno), vrednost je tipa enumeracije Keys
  - public Keys KeyCode { get; } // vraća podatak o pritisnutom tasteru bez podataka o modifikatorima
  - public Keys KeyData { get; } // vraća podatak o pritisnutom tasteru sa podacima o modifikatorima (tj. vraća vrednost prethodna dva property-ja povezana bitskim OR operatorom Modifiers | KeyCode
  - public int KeyValue { get; } // vraća celobrojnu reprezentaciju vrednosti KeyCode ((int)KeyCode)