

Tehnike obrade pogrešaka

Grafičko korisničko sučelje

2014/15.04



Standardi kodiranja - smjernice

**Samostalno proučiti materijale
u RPPP04-smjernice.pdf**

Defenzivno programiranje

❑ **Zaštita programa od neispravnog unosa podataka**

- Defenzivna vožnja automobila temelji se na načelu da vozač nikad ne može biti siguran što će učiniti drugi vozači, pa unaprijed nastoji izbjeći nezgodu za slučaj pogreške drugih vozača
- U defenzivnom programiranju ideja vodilja je da će potprogram s neispravnim podacima “opstati” i onda kada su pogreškom pozivajuće procedure predani neispravni argumenti
- Pristup “smeće unutra, smeće van” (“garbage in, garbage out”) treba zamijeniti sa: “smeće unutra, ništa van”, “smeće unutra, poruka o pogrešci van” ili “smeću zabranjen ulaz”

❑ **Osnovna pravila kojih se treba držati:**

- Provjeriti ispravnost svih vrijednosti podataka iz vanjskih izvora (datoteka, korisnik, mreža, ...)
- Provjeriti ispravnost svih vrijednosti ulaznih parametara
- Odlučiti kako postupiti u slučaju neispravnih podataka

Jednostavan primjer defenzivnog programiranja

❑ Nedefenzivno programiranje rekurzivne funkcije faktorijel

```
int faktorijel( int N )
{
    if ( N == 0 ) return 1;
    else return N * faktorijel( N-1 ) ;
}
```

- Za negativne vrijednosti nastupa (teoretski) beskonačna rekurzija.

❑ Defenzivno programiranje rekurzivne funkcije faktorijel

```
int faktorijel( int N )
{
    if ( N <= 0 ) return 1;
    else return N * faktorijel( N-1 ) ;
}
```

- Matematički netočno, ali sprječava beskonačnu rekurziju u slučaju preljeva

Tehnike obrade pogrešaka

❑ Tehnike obrade pogrešaka

- vratiti neutralnu vrijednost (0, "", NULL)
- zamijeniti neispravnu vrijednost sljedećom, moguće ispravnom
 - Npr. while (GPSfix != OK) sleep(1/100s) ...
- vratiti vrijednost vraćenu pri prethodnom pozivu npr. ret max(prev, 0)
- zamijeniti neispravnu vrijednost najbližom ispravnom, npr. min(max(kut, 0),360)
- zapisati poruku o pogrešci u datoteku, kombinirano s ostalim tehnikama
- vratiti kôd pogreške (npr. <ERRNO.H>, enum MojGrijeh) – pogrešku obrađuje neki drugi dio koda – dojava drugom dijelu:
 - postavljanjem "globalne" varijable statusa
 - extern int errno; char *sys_errlist[] ;
 - preko imena funkcije (negativna vrijednost, 0)
 - bacanjem iznimki (u nastavku)
- pozvati "globalnu" metodu za obradu pogreške, npr. perror()
- bezuvjetni završetak programa, npr. Application.Exit (CancelEventArgs)

❑ Robustnost i ispravnost (programa)

- robustnost - u slučaju pogreške omogućen je daljnji rad programa, iako to ponekad znači vratiti neispravan rezultat
- ispravnost - nikad ne vratiti neispravan rezultat, iako to značilo ne vratiti ništa

Iznimke (Exceptions)

- ❑ Iznimka predstavlja problem ili promjenu stanja koja prekida normalan tijek izvođenja naredbi programa
- ❑ U programskom jeziku C#, iznimka je objekt instanciran iz razreda koji nasljeđuje *System.Exception*

- ❑ **System.Exception – osnovni razred za iznimke**
 - StackTrace – sadrži popis poziva postupaka koji su doveli do pogreške
 - Message – sadrži opis pogreške
 - Source – sadrži ime aplikacije koja je odgovorna za pogrešku
 - TargetSite – sadrži naziv postupka koji je proizveo pogrešku

- ❑ **Iz razreda Exception izvedena su dva razreda**
 - SystemException – bazni razred za iznimke koje generira CLR
 - ApplicationException – bazni razred za iznimke aplikacije

Ugrađene iznimke - `SystemException`

❑ Neke sistemske iznimke izvedene iz razreda `SystemException`

<code>ArrayTypeMismatchException</code>	tip vrijednosti koji se pohranjuje u polje je različit od tipa polja i implicitna konverzija se ne može obaviti
<code>DivideByZeroException</code>	pokušaj dijeljenja s 0
<code>IndexOutOfRangeException</code>	indeks polja je izvan deklarirane veličine polja
<code>InvalidCastException</code>	nedozvoljena konverzija tipa
<code>OutOfMemoryException</code>	nedostatak memorije za alociranje objekta
<code>OverflowException</code>	preljev pri izračunavanju aritmetičkog izraza
<code>NullReferenceException</code>	referenci nije pridružen objekt
<code>StackOverflowException</code>	stog je prepunjen

Obrada iznimki

- ❑ Obrada iznimki sprječava nepredviđeni prekid izvođenja programa
- ❑ Iznimka se obrađuju tzv. rukovateljem iznimki (exception handler)
 - Obrada pogreške sastoji se razdvajanju kôda u blokove *try*, *catch* i *finally*

```
try {  
    //dio kôda koji može dovesti do iznimke  
}  
catch (Exception1 exOb) {  
    //kôd koji se obavlja u slučaju iznimke tipa Exception1  
}  
catch (Exception2 exOb) {  
    //kôd koji se obavlja u slučaju iznimke tipa Exception2  
}  
...// ostali catch blokovi  
finally {  
    //kôd koji se obavlja nakon izvođenja try, odnosno catch bloka
```


Postupak obrade iznimki

❑ Varijante obrade

- Za jedan try blok može postojati jedan ili više catch blokova koji obrađuju različite vrste pogrešaka.
- Kada dođe do pogreške u try bloku, a postoji više catch blokova, obavlja se onaj catch blok koji obrađuje nastali tip iznimke. Ostali catch blokovi neće se obaviti.
- Ako postoji više catch blokova, posljednji se navodi blok koji obrađuje općenite iznimke (tipa System.Exception)

❑ Primjer: Kodiranje\TryCatch

Generiranje (bacanje) iznimke naredbom throw

❑ Sintaksa

- `throw exceptOb;`
 - `exceptOb` – objekt tipa `Exception` ili iz njega izvedenih razreda
 - `npr.throw new ApplicationException("Nova iznimka");`
- `throw;`
 - prosljeđivanje iznimke na vanjski blok (rethrow)

❑ Primjer, Kodiranje\ExceptionInfo

- bacanje iznimki i prikaz informacije o iznimci

```
throw new ApplicationException("poruka");  
...  
Console.WriteLine( "Izvor: {0}", e.Source );  
Console.WriteLine( "Postupak: {0}", e.TargetSite );  
Console.WriteLine( "Poruka: {0}", e.Message );  
Console.WriteLine( "Trag: {0}", e.StackTrace );
```

❑ Primjer: Kodiranje\Rethrow

- prosljeđivanje uhvaćene iznimke

Kreiranje vlastitih iznimki

- ❑ Definiranjem razreda izvedenog iz razreda `ApplicationException`
- ❑ Primjer:  `Kodiranje\CustomException`

- `catch (Iznimka e)`

```
class Iznimka : ApplicationException
{
    private string val;
    public Iznimka(string str) : base(str)
    {
        val = str;
    }
    public override string Message
    {
        get
        {
            return "Nije neparan " + val;
        }
    }
}
```

Preporuke za korištenje iznimki

- Bacati iznimke samo u stanjima koja su stvarno iznimna
- Izbjegavati prazne blokove za hvatanje iznimki (“catch { }”)
- Izbjegavati hvatanje osnovne iznimke `Exception`,
 - koristiti specifične iznimke, `SystemException` ili `ApplicationException`, znajući što bacaju vlastite knjižnice
- U poruci iznimke uključiti sve informacije o kontekstu nastanka iznimke
- Zapisivati trag bačenih iznimki (log)
- Prosljeđivanje iznimki raditi samo kada želimo specijalizirati iznimku
- Koristiti iznimke za obavijest drugim dijelovima programa o pogreškama koje se ne smiju zanemariti
- Ne bacati iznimke za pogreške koje se mogu obraditi lokalno
- Izbjegavati bacanje iznimki u konstruktorima i destruktorima, osim ako ih na istom mjestu i ne hvatamo
- Razne tipove pogrešaka obrađivati na konzistentan način kroz čitav kod
- Razmotriti izradu centraliziranog sustava za dojavu iznimki u kodu


Primjer centralizirane obrade iznimki

❑ Primjer: Kodiranje\CentraliziraniSustav

```
public static void ReportError(Exception iznimka) {
    Console.WriteLine("Neocekivana iznimka:  " + iznimka.Message);
    Console.WriteLine("Mjesto iznimke:");
    Console.WriteLine(iznimka.StackTrace);
    Console.WriteLine("\nPritisni tipku...");
    Console.ReadLine();
}

...
try {
    Exception ex = new ApplicationException("\nPoruka iznimke...\n");
    throw ex;
}
catch (Exception ANYexception)
{
    ReportError(ANYexception);
}
```

Unutarnje iznimke

- ❑ `Exception.InnerException` – dobavlja instancu razreda *Exception* koja je izazvala aktualnu iznimku
- ❑ Primjer:  `Kodiranje\InnerException`

```
class Primjer
{
    public void F()
...
        throw new ApplicationException
            ("Iznimka u Primjer.F() :", e);
...
class Program
{
    static void Main(string[] args)
    {
        catch (Exception e)
        {
            Console.WriteLine("Iznimka u Main: "
                + "{0}\nInnerException: {1}",
                e.Message, e.InnerException.Message);
        }
    }
}
```

Ostale mogućnosti obrade iznimki

- ❑ **Eksplicitno postavljanje provjere preljeva za operacije s cijelim brojevima (integral-type arithmetic operations and conversions)**
 - checked context – aritmetički preljev podiže iznimku
 - unchecked context – aritmetički preljev bude ignoriran, a rezultat "odrezan" (truncated)
 - Ako nije ništa navedeno, pretpostavlja se da je *unchecked*

❑ **Primjer:** **Kodiranje\CheckedUnchecked**

```
int deset = 10;

// bez provjere
i = unchecked (2147483647 + deset); // negativno

// s provjerom
i = checked (2147483647 + deset); // iznimka
```

Tvrdnje

❑ Tvrdnja (Assert)

- logička (Boolean) naredba (metoda u C#) kojom se program testira tako da njen uvjet mora biti istinit, a ukoliko nije, program pada
- tvrdnje se koriste za uklanjanje pogrešaka (debugging) i dokumentiranje ispravnog rada programa
- koriste se u fazi kodiranja, naročito u razvoju velikih, kompliciranih programa, te u razvoju programa od kojih se zahtijeva visoka pouzdanost
- omogućavaju programu da sam sebe provjerava tijekom izvođenja
- pišemo ih na mjestima gdje se pogreške ne smiju pojaviti

❑ Prostor imena **System.Diagnostics**, naredba **Debug.Assert**

- Logički izraz za koji se pretpostavlja (tvrdi) da je istinit
- Poruka koja se ispisuje ako izraz nije istinit

❑ Primjer: 📁 Kodiranje\Assert

```
int brzina = 650;  
Debug.Assert((0 <= brzina) && (brzina <= 600),  
             "Neispravna brzina.");
```


Preporuke za korištenje tvrdnji

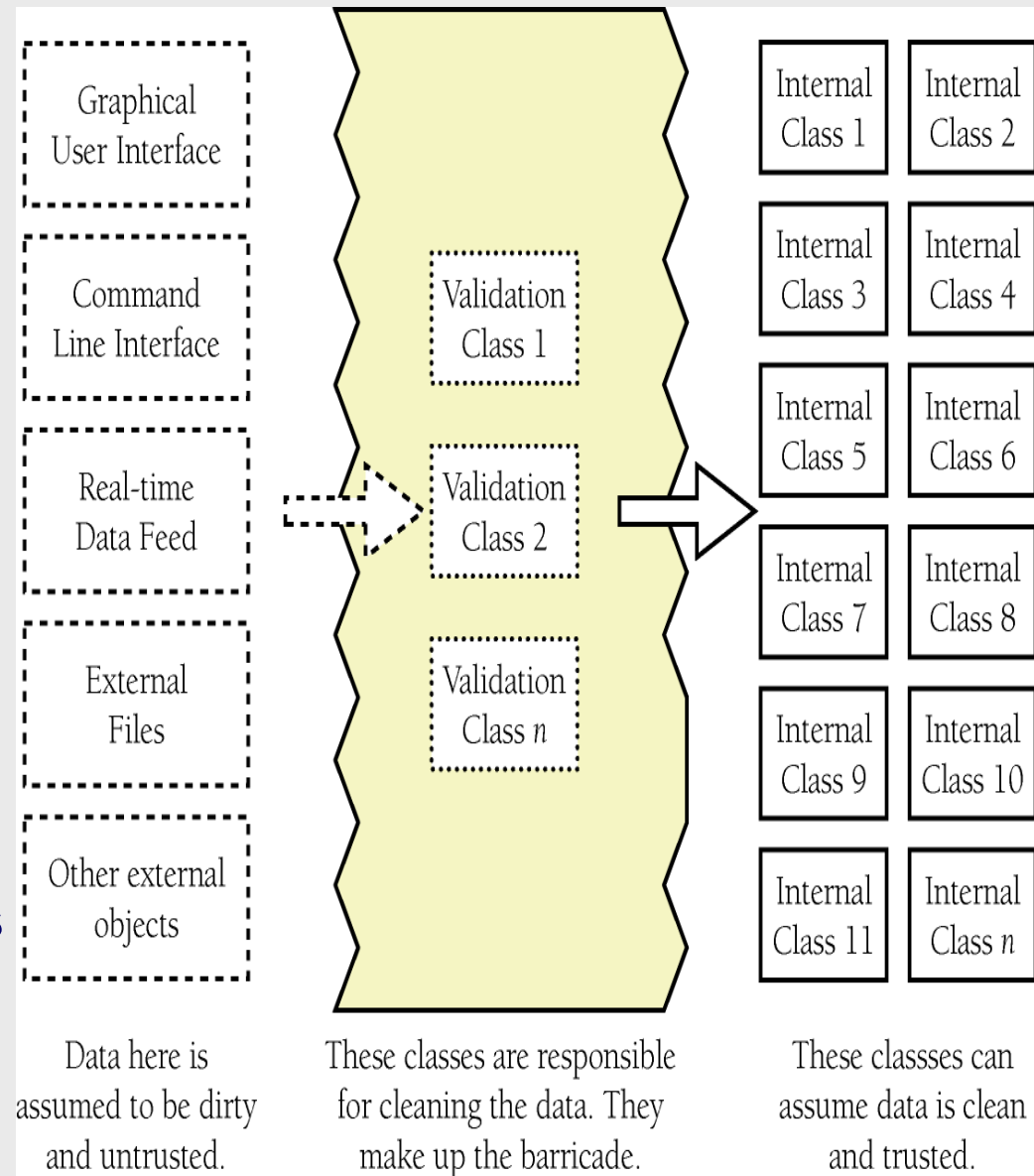
- ❑ Obradu pogreške (iznimke) pisati tamo gdje očekujemo pogreške
- ❑ Koristiti tvrdnje tamo gdje nikad ne očekujemo pogreške
- ❑ Za jako robustan kod koristiti tvrdnje + kod za obradu pogreške

- ❑ Izbjegavati poziv metoda u izrazima tvrdnji
 - `npr. Debug.Assert (Obavi(), "Neobavljeno");`

- ❑ Koristiti tvrdnje
 - za dokumentiranje i verificiranje uvjeta koji moraju vrijediti prije pozivanja metode ili instanciranja razreda ("preconditions"), te
 - uvjeta koji moraju vrijediti poslije djelovanja metode ili rada s razredom ("postconditions").

Koncept barikada

- ❑ Konstrukcija sučelja kao granica prema “sigurnim” dijelovima koda
- ❑ Definiranje dijelova softvera koji će rukovati “prljavim” (nesigurnim) podacima i drugih koji rukuju samo s “čistim” podacima (validacijskih razreda koji su odgovorni za provjeru ispravnosti podataka i sačinjavaju barikadu, te internih razreda koji rukuju s podacima za koje se pretpostavlja da su provjereni i ispravni)



Koncept barikada

❏ Primjer: 📁 Kodiranje\Barikade

```
class Primjer
{
    private int index; //privatna varijabla (iza barikade)
    public Primjer()
    {
    }
    public int Index //javna metoda (služi kao sučelje barikade)
    {
        get { return index; }
        set
        {
            //provjeravamo je li unutar zadanih granica,
            //ako ne, pridjeljujemo najbližu vrijednost
            if (value <= 0) { this.index = 0; }
            else if (value > 100) { this.index = 100; }
            else { this.index = value; }
        }
    }
}
```

Preporuke za korištenje barikada

- ❑ **Barikade naglašavaju razliku između tvrdnji i obrade iznimaka**
 - Metode s vanjske strane barikade trebaju koristiti kôd za obradu pogreške
 - Unutarnje metode mogu koristiti tvrdnje jer se ovdje pogreške ne očekuju!


- ❑ **Na razini razreda**
 - javne metode rukuju s “prljavim” podacima i “čiste” ih
 - privatne metode rukuju samo s “čistim” podacima.

- ❑ **Pretvarati podatke u ispravan tip odmah pri unosu**

Otkrivanje pogrešaka


- ❑ **Uobičajena zabluda programera je da se ograničenja koja se odnose na konačnu verziju softvera odnose i na razvojnu verziju**
 - Treba biti spreman žrtvovati brzinu i resurse tokom razvoja u zamjenu za olakšani razvoj
- ❑ **Ofenzivno programiranje – učiniti pogreške u fazi razvoja toliko očitim i bolnim da ih je nemoguće zanemariti**
 - osigurati da `assert` naredbe uzrokuju prekid izvođenja pri pogrešci
 - popuniti bilo koju alociranu memoriju prije upotrebe radi detektiranja eventualnih problema s njenom alokacijom
 - popuniti alocirane datoteke ili tokove podataka prije upotrebe radi detektiranja eventualnih grešaka u formatu datoteka ili podataka
 - osigurati da svaka `case` naredba koja propagira do `default` slučaja uzrokuje pogrešku koju nije moguće zanemariti
 - napuniti objekt “smećem” (junk data) neposredno prije njegovog brisanja
- ❑ **Napad je najbolja obrana**
 - pogriješiti toliko jako tijekom razvoja da ne predstavlja problem u pogonu

Otkrivanje pogrešaka (2)

- ❑ **Planirati uklanjanje dijelova programa koji služe kao pomoć u otkrivanju pogrešaka u konačnoj verziji softvera**
 - koristiti alate za upravljanje verzijama
 - koristiti ugrađene predprocesore za uključivanje/isključivanje dijelova koda u pojedinoj verziji
 - korištenje vlastitog (samostalno napisanog) predprocesora
 - zamjena metoda za otkrivanje pogrešaka u konačnoj verziji “praznim” metodama koje samo vraćaju kontrolu pozivatelju
- ❑ **Primjer:  Kodiranje\Debug**

```
#define RAZVOJ //definiramo simbol
...
#if (RAZVOJ)
    // kod za debugiranje
    Console.WriteLine("Poruka UNUTAR koda za debugiranje!");
#endif
...
Console.WriteLine("Poruka IZVAN koda za debugiranje!");
```

Otkrivanje pogrešaka (3)

- ❑ Umjesto `#if` i `#endif` koristiti ***Conditional*** (iz *System.Diagnostics*)
- ❑ Kod za testiranje odvojiti u posebni postupak i iznad postupka navesti atribut ***Conditional***
 - U slučaju da uvjet nije zadovoljen, u kompiliranoj verziji nije uključen poziv označenog postupka
 - Simbol se može definirati u kodu, ali i kao parametar prilikom kompiliranja
 - Properties → Build → Conditional compilation symbols
- ❑ **Primjer:**  **Kodiranje\Debug**

```
#define RAZVOJ //definiramo simbol
...
Test();
...
[Conditional("RAZVOJ")]
static void Test()
{
    Console.WriteLine("Poruka iz postupka Test");
}
```

Količina defenzivnog koda u završnoj verziji

- ☐ **Ostaviti kôd koji radi provjere na opasne pogreške**
- ☐ **Ukloniti kôd koji provjerava pogreške s trivijalnim posljedicama**
 - Ukloniti pretprocesorskim naredbama, a ne fizički
- ☐ **Ukloniti kôd koji može uzrokovati pad programa**
 - U konačnoj verziji treba omogućiti korisnicima da sačuvaju svoj rad prije nego se program sruši.
- ☐ **Ostaviti kôd koji u slučaju pogreške omogućava “elegantno” rušenje programa**
- ☐ **Ostaviti kôd koji zapisuje pogreške koje se događaju pri izvođenju**
 - Zapisivati poruke o pogreškama u datoteku.
- ☐ **Treba biti siguran da su sve poruke o pogreškama koje softver dojavljuje “prijateljske”**
 - Obavijestiti korisnika o “unutarnjoj pogrešci” i navesti e-mail ili broj telefona tako da korisnik ima mogućnost prijaviti pogrešku


Zadaci za vježbu

- ❑ **Obraditi iznimku kada se ostvari Debug.Assert**

- ❑ **Implementirati razred Niz koji osim niza cijelih brojeva sadržava gornju i donju granicu niza.**
 - Baciti iznimku ako je gornja granica manja od donje, te ako se preko indeksa pristupa članu izvan gornje odnosno donje granice.
 - Napisati vlastiti razred za obradu navedenih iznimki.

- ❑ **Implementirati razred Temperature koji prima razred Niz. Razred treba imati metodu za računanje prosječne temperature.**
 - Ubaciti provjere da niz temperatura nije prazan prije izračuna.
 - Provjeriti i da je dobiveni prosjek unutar granica definirane gornje i donje vrijednosti niza.

Životni vijek objekta

- ❑ **Primjer:**  **Kodiranje\Using**
- ❑ **Instancirani objekt postoji dok ga sakupljač smeća (GarbageCollector) ne uništi**
 - GC će ga obrisati ako na njega ne pokazuje ni jedna referenca
- ❑ **Što ako ne možemo čekati GC?**
 - Implementirati sučelje **IDisposable** (postupak **Dispose**)

```
public class Razred : IDisposable {  
    public void Dispose() {  
        //zatvaranje datoteke, konekcije  
        // i sličnih "dragocjenih" resursa  
        ...  
    }  
}
```

- Ako neki razred implementira **IDisposable**, preporuka je da se za objekte tog razreda **Dispose** uvijek pozove nakon što objekt više ne bude potreban.

IDisposable, using blok i iznimke

- ❑ **Dispose se može pozvati eksplicitno**
- ❑ **Što ako se dogodi iznimka prije poziva postupka Dispose?**
 - Koristiti tzv. *using blokove*
 - Za objekt stvoren unutar *using bloka*, Dispose se automatski poziva nakon napuštanja bloka (bez obzira na razlog izlaska iz bloka)

```
Razred r1 = new Razred("A1");  
using (Razred r2 = new Razred("B2")) {  
    Razred r3 = new Razred("C3");  
    r3 = null; //što se izaziva ovom naredbom?  
    GC.Collect(); // što ako nije pozvan GC?  
    GC.WaitForPendingFinalizers();  
    throw new ApplicationException("Poruka");  
}  
r1.Dispose();
```

- Using blok se može koristiti samo za one razrede koji implementiraju **IDisposable**

Grafičko korisničko sučelje

2014/15.05

Grafičko korisničko sučelje

❑ Graphical User Interface (GUI)

❑ *WindowsForms, WPF, WebForms*

- *WindowsForms* - namijenjene izradi klijentskih aplikacija
 - alternativa (nasljednik) *WPF - Windows Presentation Foundation*
- *WebForms* - namijenjene izradi Web stranica

❑ `System.Windows.Forms` prostor imena za razrede Windows GUI

- Skup baznih razreda za podršku prozorima i kontrolama sučelja
- Svaka forma nasljeđuje `System.Windows.Forms`

❑ Forma (*form*)

- strukturirani prozor ili okvir koji sadrži prezentacijske elemente za prikaz i unos podataka

❑ Forma i prozor (*window*)

- Svaka forma je prozor, ali se uobičajeno podrazumijeva da je forma dijalog na kojem se nalazi određeni broj kontrola

Hijerarhija razreda

`System.Object`

`System.MarshalByRefObject`

`System.ComponentModel.Component`

`System.Windows.Forms.Control`

`System.Windows.Forms.ScrollableControl`

`System.Windows.Forms.ContainerControl`

`System.Windows.Forms.Form`

☐ **MarshalByRefObject**

- automatizira prenošenje objekata putem proxy-a preko različitih aplikacijskih domena (.NET Remoting)

☐ **Component**

- razred koji implementira sučelje ***IComponent***
- omogućava dijeljenje objekata između aplikacija
- nema vidljivih dijelova

☐ **Control**

- osnovni bazni razred za kontrole s vizualnim sučeljem (npr. gumb)

☐ **ScrollableControl**

- osnovni razred za kontrole koje podržavaju auto-scrolling

☐ **ContainerControl**

- funkcionalnost potrebna da bi kontrola sadržavala druge kontrole

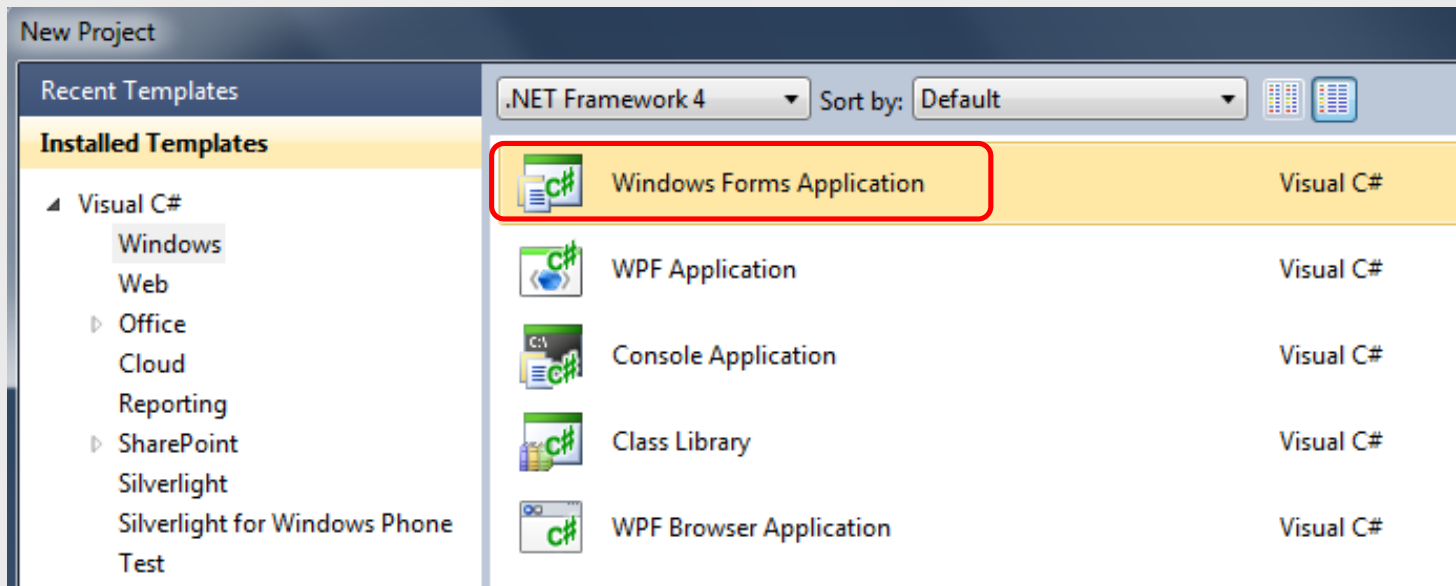
☐ **Form**

- razred koji predstavlja (praznu) formu ili dijalog kutiju i može sadržavati druge kontrole
- iz ovog razreda se najčešće izvodi razred u kojem se zatim implementiraju potrebne specifičnosti

Stvaranje nove Windows Forms aplikacije

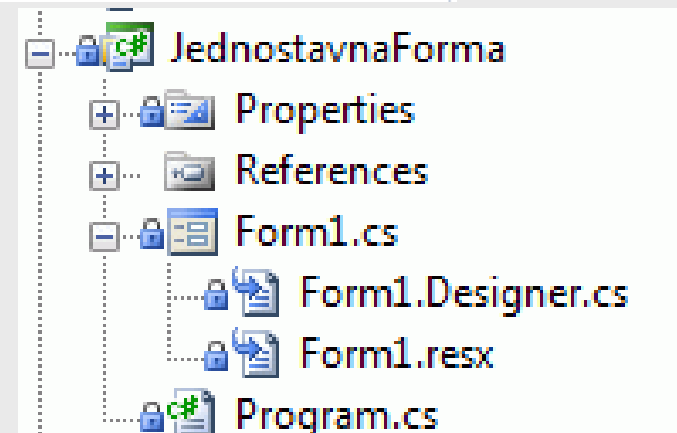
□ Primjer: GUI\JednostavnaForma

- File→New→Project – u odjeljku Windows odabrati Windows Forms Application



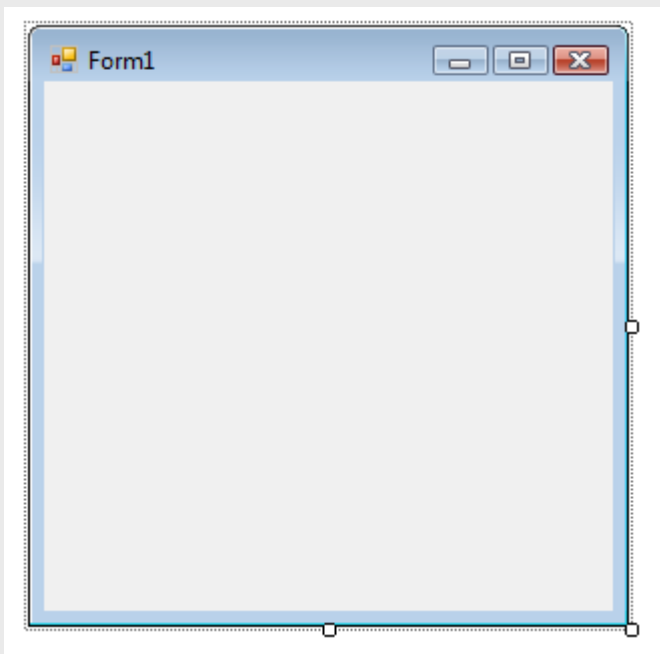
□ Stvara se projekt s nekoliko dokumenata:

- Form1.cs (programski kôd forme)
 - Form1.Designer.cs (definicija izgleda)
 - Form1.resx (resursi)
 - nastane nakon dodavanja kontrola
- Program.cs (glavni program)



Izgled forme

❑ Form1.cs[Design] i Form1.cs



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace JednostavnaForma
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

partial class – ostali
dijelovi razreda nalaze se u
drugim datotekama (npr.
Form1.designer.cs na
sljedećem slajdu)

Automatski generirani kod u formi

❑ Inicijalizacija: `InitializeComponent()`

- svako postavljanje svojstva forme ili postavljanje kontrole na formu u razvojnoj okolini generira kôd unutar funkcije `InitializeComponent()`
- nestručne ručne izmjene tijela `InitializeComponent` mogu uzrokovati da *Visual Designer* više ne bude u stanju sinkronizirati kôd i izgled forme

```
partial class Form1
{
    ...
    private void InitializeComponent()
    {
        this.SuspendLayout();
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(284, 261);
        this.Name = "Form1";
        this.Text = "Form1";
        this.ResumeLayout(false);    }
}
```

Pokretanje forme

❑ Program.cs

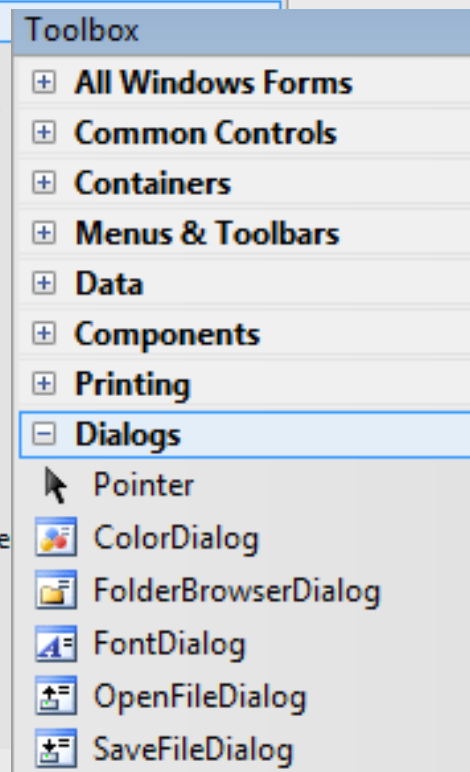
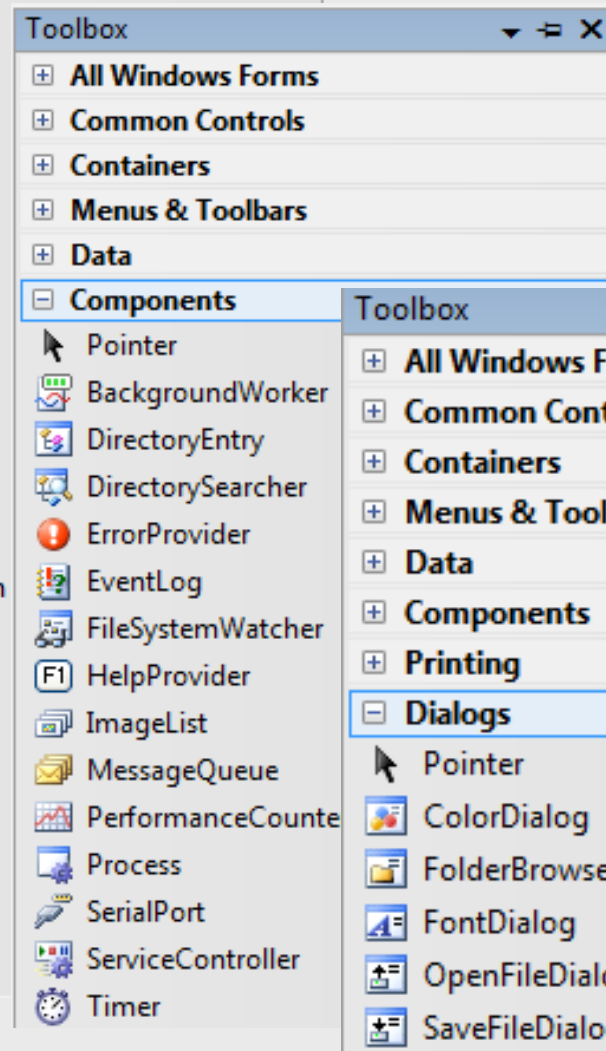
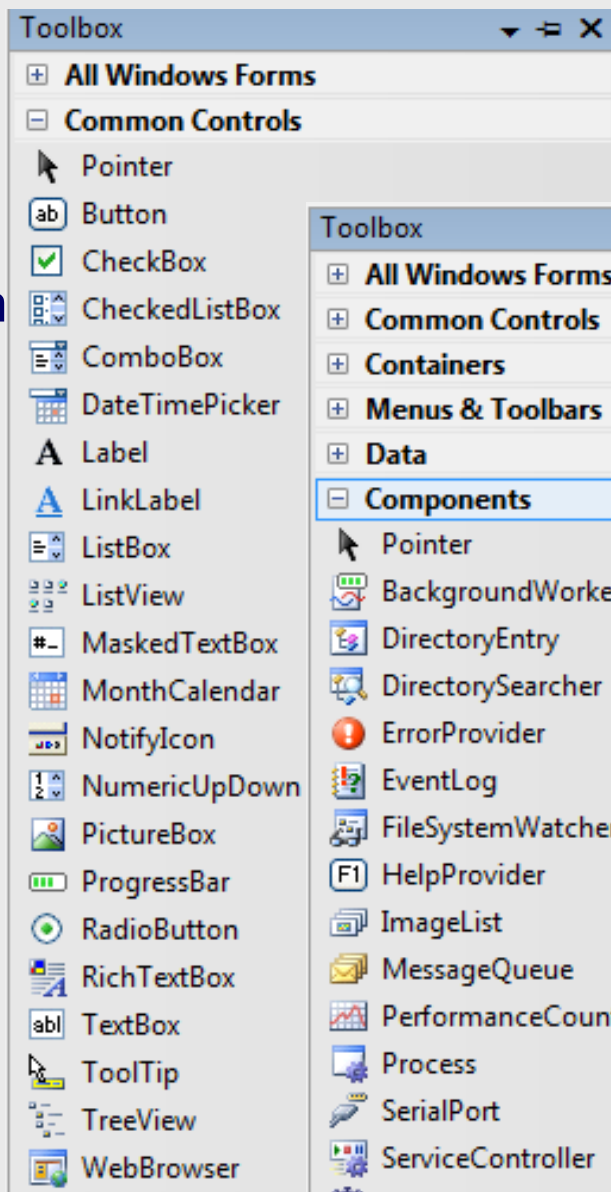
- Aplikacija kreće pokretanjem statičke funkcije *Main()*
- U *Main()* instanciramo i pokrenemo formu
- Grafičko sučelje dalje je vođeno događajima koji se zbivaju nad formom i objektima forme

❑ Primjer: izvođenje programa naredbu po naredbu

```
static class Program
{
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
}
```

Elementi grafičkog sučelja

- ❑ **Kontrole, komponente, dijalozi**
- ❑ **Dodajemo ih dovlačenjem (*drag-drop*) iz kutije s alatima (*Toolbox*)**
- ❑ **Dizajn forme i kontrola**
 - označavanje jedne ili više kontrola mišem, a zatim premještanje, razvlačenje ili smanjivanje
 - Izbornik *Format*



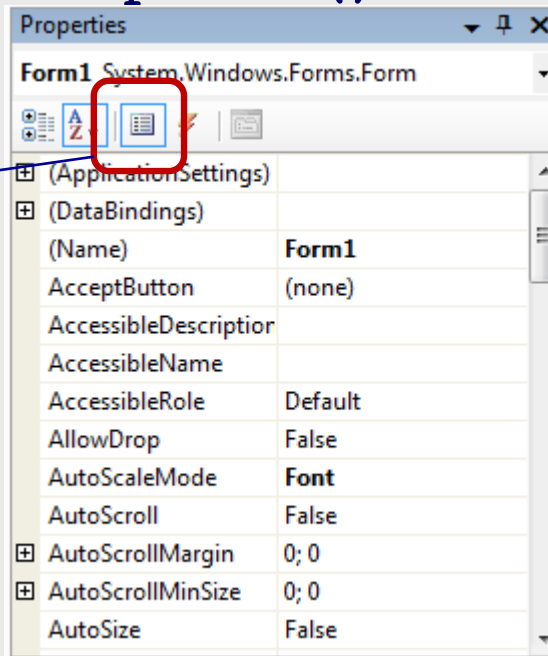
Svojstva i događaji

❑ Svojstva objekata (forme i ostalih kontrola)

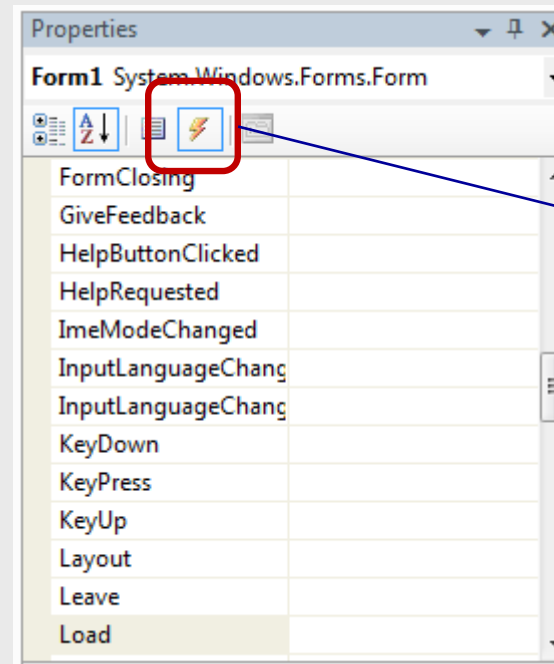
- inicijalno se postavljaju prilikom dizajna (u *Property Window*), označavanjem objekta i izbornikom *View – Properties Window* ili desnim klikom + *Properties*
- neka svojstva mogu biti i dinamički promijenjena, pri izvođenju programa

❑ Za svojstva početno postavljena u dizajnu generira se kod u `InitializeComponent()`

Svojstva



Događaji



Vrste (razredi) kontrola

- ☐ **Button** (gumb) – osnovna namjena je pokretanje akcija
- ☐ **Label** – prikazuje tekst, bez mogućnosti unosa

- ☐ **TextBox** – prikazuje tekst koji se može unositi
- ☐ **CheckBox** – predstavlja varijablu s dva stanja (true/false)
- ☐ **RadioButton** – predstavlja grupu da/ne izbora, stavlja ih se više u isti spremnik, a samo jedan gumb bude potvrđan

- ☐ **ListBox** i **CheckedListBox** – lista stavki, odabire se jedna ili više
- ☐ **ComboBox** – padajuća lista u kojoj se odabire samo jedna stavka

- ☐ **LinkLabel** – prikazuje jednu ili više hiperveza (hyperlink)
- ☐ **PictureBox** – prikazuje sliku

- ☐ **DataGridView, ListView, TreeView** – kontrole za prikaz kolekcije podataka

- ☐ **GroupBox i Panel** – spremnici kontrola
- ☐ ...

Standardna svojstva i postupci kontrola

❑ Standardna svojstva i postupci

Svojstva	
BackColor	Boja pozadine
BackgroundImage	Slika pozadine
Enabled	Omogućena aktivnost (događaji)
Focused	Kontrola ima fokus (trenutno se koristi)
Font	Font svojstva Text
ForeColor	Boja prednjeg plana. Uobičajeno se odnosi na svojstvo Text
TabIndex	Redni broj kontrole, određuje redoslijed kojim ona dolazi u fokus (npr. tipkom Tab)
TabStop	Oznaka da kontrola dolazi u fokus postavljenim redoslijedom
Text	Tekst koji se prikazuje na kontroli
TextAlign	Poravnanje, horizontalno (left, center, right), vertikalno (top, middle or bottom).
Visible	Oznaka da je kontrola vidljiva
Postupci	
Focus	Prijenos fokusa na kontrolu
Hide	Skrivanje kontrole (postavlja Visible na false).
Show	Pojavljivanje kontrole (postavlja Visible na true).

Razred *Form*

□ Svojstva

Text	naslov forme
BackColor, ForeColor	boja pozadine i prednjeg plana
Font	koji font se koristi prilikom rada na formi
WindowState	stanja (Normal, Maximized, Minimized)
Size	dimenzije
Location, DesktopLocation	postavljanje pozicije forme na ekranu
MaximizeBox, MinimizeBox ControlBox	elementi upravljanja prozorom
FormBorderStyle	Fixed3D, FixedDialog, FixedSingle, ...
TopMost	da li će forma uvijek biti vidljiva (na vrhu)

□ Događaji

Activated	dogaća se kad forma dobije fokus
Deactivated	poziva se kad forma gubi fokus
Closing	poziva se prilikom zatvaranja forme (prije samog zatvaranja)
Closed	poziva se prilikom zatvaranja forme (nakon samog zatvaranja)
Load	poziva se prilikom prvog učitavanja forme

Svojstva forme

❑ Postupci načina prikaza

- `Show` – nemodalni prikaz
 - moguće je kliknuti na neku od više otvorenih formi
- `ShowDialog` – modalni prikaz
 - forma na vrhu ne dozvoljava da se aktiviraju donje
- `Hide` – sakriva formu

❑ Svojstva prikaza

- `Size` – veličina prozora
- `Location` – položaj lijevog-gornjeg kuta prozora u odnosu na ekran
- `StartPosition` – početni položaj forme pri izvođenju (logički)
- `DesktopLocation` – položaj prozora u odnosu na radnu površinu (ovisno o smještaju *taskbara*)

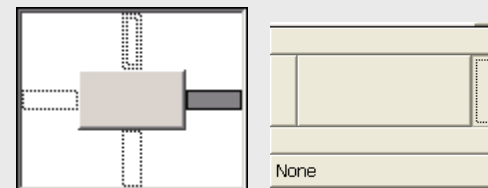
❑ Događaji

- `LocationChanged`
- `SizeChanged`



Svojstva za dinamičko razmještanje kontrola

❑ Veličina, položaj i razmještaj u pogonu



❑ Zadatak za vježbu

- Proučiti svojstva za dinamičko razmještanje kontrola (`Anchor`, `Dock`, `DockPadding`, `Location`, `Size`, `MinimumSize`, `MaximumSize`).

Anchor	sidrenje rubova kontrole prema rubu spremnika: Top, Left, Bottom, Right (fiksira udaljenost kontrole od spremnika)
Dock	Automatsko prislanjanje kontrole na rub spremnika: Top, Left, Bottom, Right, Fill (Center), None po mogućnosti kontrola mijenja veličinu
DockPadding	Razmak od ruba (samo za spremnike), standardna vrijednost = 0.
Location	Položaj gornjeg-lijevog kuta kontrole, relativno u odnosu na spremnik.
Size	Veličina: Size struktura sa svojstvima Height i Width .
MinimumSize	Minimalna veličina forme
MaximumSize	Maksimalna veličina forme

Događaji

❑ Događaj - mehanizam razreda za dojavu zbivanja nad objektom

- primjeri događaja: klik mišem na gumb, unos teksta u kućicu formulara, učitavanje/zatvaranje forme, ...
- povećava se modularnost programa
- osim na GUI, koriste se i šire

❑ Rukovatelj događajem (*Event handler*)

- Postupak koji obrađuje događaje
- Uobičajenog naziva *ControlName_EventName*
- Uobičajeni argumenti su objekt događaja i podaci događaja

❑ Primjer: GUI\JednostavnaForma

- kako debugirati izvođenje događaja ?
- što kada se promijeni naziv objekta (npr. *button1* postane *gumb*) ?

```
private void button1_Click(object sender, EventArgs e)
{
    this.Text = "Pozdrav!";
    label1.Text = System.DateTime.Now.ToString();
}
```

Događaji i delegati

❑ Tip delegata

- opisuje kako izgleda postupak (za npr. obradu događaja)
- objekt ovog razreda ne sadrži podatke, već reference na jedan ili više postupaka (slično pokazivaču na funkciju u C/C++)
- postupci se registrišu na listi poziva delegata objekta
 - Pridruživanje jednog postupka delegatu =
 - Dodavanje postupka delegatu +=
 - Uklanjanje nekog postupka iz delegata -=

❑ Događaj je delegat kojem su dozvoljene samo operacije += i -=

- Više o delegatima i događajima u primjeru s vlastitom kontrolom

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

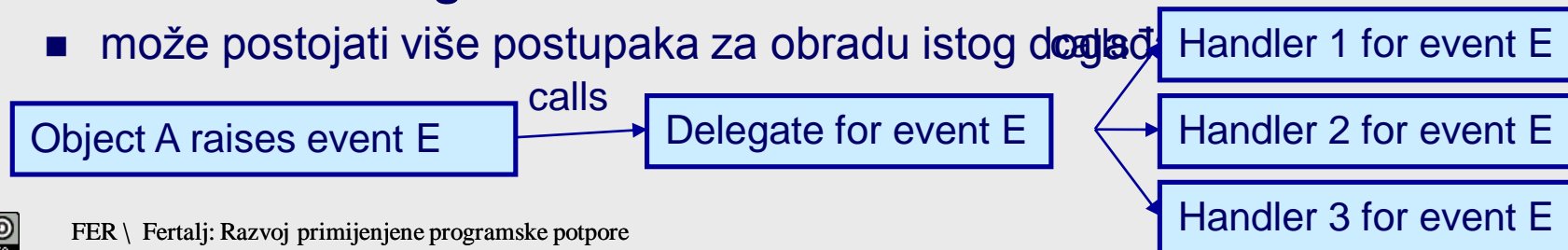
Događaj

Tip delegata

Event handler

❑ Event multicasting

- može postojati više postupaka za obradu istog događaja



Načini registracije postupka obrade događaja

❑ Klasično, instanciranjem delegata

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

Događaj

Tip delegata

Event handler

❑ Navođenjem naziva postupka obrade događaja objekta

```
this.button1.Click += this.button1_Click;
```

❑ Korištenjem ključne riječi *delegate* i anonimne metode

- parametri se mogu ispustiti

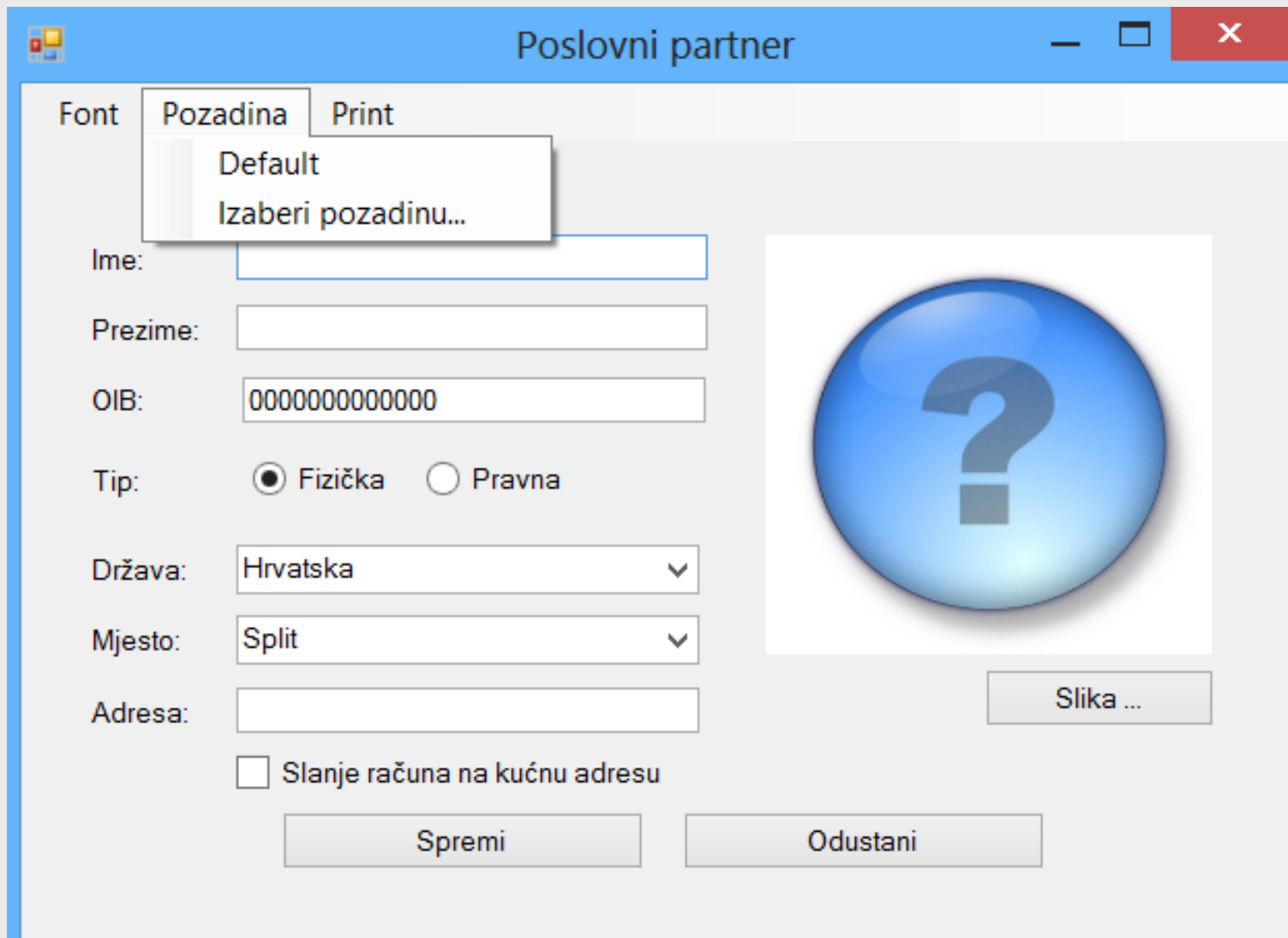
```
this.button1.Click += delegate(object sender, EventArgs e) {  
    //kod za obradu događaja  
};
```

❑ Pisanjem lambda izraza

```
this.button1.Click += (s, e) => {  
    //kod za obradu događaja  
};
```

Izrada zaslonske maske za unos podataka

❑ Primjer: GUNPartner



Poslovni partner

Font Pozadina Print

Default
Izaberi pozadinu...

Ime:

Prezime:

OIB:

Tip: ☒ Fizička ☐ Pravna

Država:

Mjesto:

Adresa:

☐ Slanje računa na kućnu adresu

Slika ...

Spremi Odustani

Button, Label i TextBox

❑ Button

Svojstva	
FlatStyle	Plošni izgled
Image	Slika – ako nije postavljena, gumb prikazuje vrijednost svojstva Text
ImageList	Lista slika, od kojih jedna može u nekom trenutku biti prikazana
ImageIndex	Indeks trenutno prikazane slike
Događaji	
Click	Klik mišem ili <i>Enter</i> kada je kontrola u fokusu

❑ Label i TextBox

Svojstva	
AcceptsReturn	<i>True</i> – <i>Enter</i> umeće znak za novi redak
Multiline	prikaz teksta koji ima više redaka (uobičajeno se postavlja i ScrollBars)
PasswordChar	Znak koji će se pojavljivati pri unosu umjesto teksta koji se unosi. Ako se izostavi prikazuje se tekst koji se unosi
ReadOnly	<i>True</i> – pojavljuje se siva pozadina i ne dozvoljava unos
ScrollBars	Vrsta kliznih traka za <i>Multiline</i> : <i>none</i> , <i>horizontal</i> , <i>vertical</i> , <i>both</i> .
Text	Tekst
Događaji	
TextChanged	Promjena teksta (za svaki znak)

RadioButton, GroupBox i Panel

❑ **RadioButton**

- uobičajeno predstavlja grupu da/ne izbora
- kada se postavi u spremnik može se označiti samo jedna instanca kontrole

Svojstva	
Appearance	<i>Normal</i> ili <i>Button</i>
Checked	<i>true</i> ako je kontrola označena, inače <i>false</i>
AutoCheck	<i>true</i> : kontrola postavlja oznaku sukladno vrijednosti <i>Checked</i>
Događaji	
AppearanceChanged	promjena svojstva <i>Appearance</i>
CheckedChanged	promjena stanja <i>Checked</i>

❑ **GroupBox i Panel – spremnici kontrola s labelom**

- `Controls` – lista sadržanih kontrola
- `BorderStyle` – obrub (*Fixed3D*, *FixedSingle*, *None*)
- `AutoScroll` – automatska pojava kliznika kad se ne vide kontrole (*Panel*)
- `Text` – labela na vrhu *GroupBox* kontrole (samo *GroupBox*)

❑ **Primjer, premještanje panela u dizajnu**

ListBox, CheckedListBox, ComboBox

- ❑ **ListBox i CheckedListBox** – lista stavki, odabire se jedna ili više
- ❑ **ComboBox** – padajuća lista u kojoj se odabire samo jedna stavka

Svojstva	
Items	<i>ListBoxItems.ObjectCollection</i> objekt s listom svih stavki
MultiColumn	Prikaz stavki u više stupaca (samo ListBox)
SelectedIndex	indeks selektirane stavke ili -1 kada nijedna nije odabrana
SelectedItem	Object referenca na selektiranu stavku
Sorted	True: stavke su sortirane
Postupci	
Add, Clear, Insert, Remove, ...	Rukovanje stavkama (sjetimo se kolekcija)
ClearSelected	briše sve slektirane stavke
FindString,	nalazi prvu stavku u listi koja počinje sa zadanim stringom
FindStringExact	nalazi prvu stavku u listi koja točno odgovara zadanom stringu
Sort	sortira stavke u listi
Događaji	
SelectedIndexChanged	podizhe se kad se promijeni SelectedIndex

Padajuće liste

❑ Padajuća lista (**ComboBox**): Mjesto i Država

- Kada odaberemo državu želimo ponuđene samo gradove te države

```
listaDrzava[0] = "Hrvatska";  
listaDrzava[1] = "Njemačka";  
...  
comboBoxDrzava.DataSource = listaDrzava; // ekvivalent Items u dizajnu  
  
listaGradova[0] = new string[] { "Split", "Zagreb", "Dubrovnik" };  
listaGradova[1] = new string[] { "Koeln", "Berlin", "Frankfurt"};  
...  
comboBoxMjesto.DataSource =  
    listaGradova[comboBoxDrzava.SelectedIndex];
```

```
private void comboBoxDrzava_SelectedIndexChanged(  
    object sender, EventArgs e)  
{  
    comboBoxMjesto.DataSource =  
        listaGradova[comboBoxDrzava.SelectedIndex];  
}
```

PictureBox

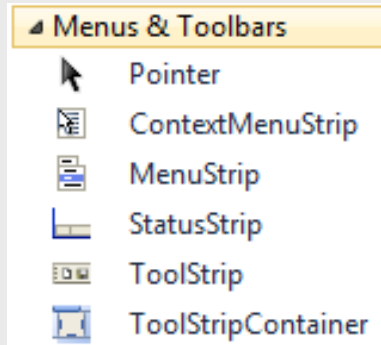
❑ Neka od svojstava

- `Image` – slika (tipa *Image*)
- `ImageLocation` – putanja do slike na disku
- `BorderStyle` – obrub (`Fixed3D`, `FixedSingle`, `None`)
- `SizeMode` – veličina i položaj bitmape (`enum PictureBoxSizeMode`)
 - `Normal` (default) – slika u gornjem lijevom kutu kontrole
 - vidi se samo dio slike ako je slika veća od kontrole
 - `CenterImage` – slika u sredini kontrole
 - vidi se samo dio slike ako je slika veća od kontrole
 - `StretchImage` – prilagodba veličine slike prema veličini kontrole
 - `AutoSize` – prilagodba veličine kontrole prema veličini slike
 - `Zoom` – promjena veličine kontrole uz zadržavanje omjera visina:širina

❑ Neki od postupaka

- `Load` – učitava sliku s određenog url-a

Izbornici



- ❑ **MenuStrip** – korijen sustava izbornika
- ❑ **ToolStripMenuItem** – stavka izbornika

Svojstva	
Checked	da li se kraj stavke nalazi kvačica
Text	tekst stavke, vrijednost "-" prikazuje separator
ShortcutKeys	definira tipkovničku kraticu za pozivanje stavke
Događaji	
Click	podigne se kad se klikne na stavku ili utipka tipkovničku kraticu
Postupci	
GetCurrentParent	vraća <i>ToolStrip</i> objekt kojeg je dio ta stavka
PerformClick	generira <i>Click</i> događaj nad stavkom (simulira akciju korisnika)

- ❑ **ToolStrip** – traka s alatima
- ❑ **ContextMenuStrip** – izbornik zavisan o kontekstu
 - Desni klik na kontrolu koja ima definirano svojstvo `ContextMenuStrip`
- ❑ **StatusStrip** – statusna traka

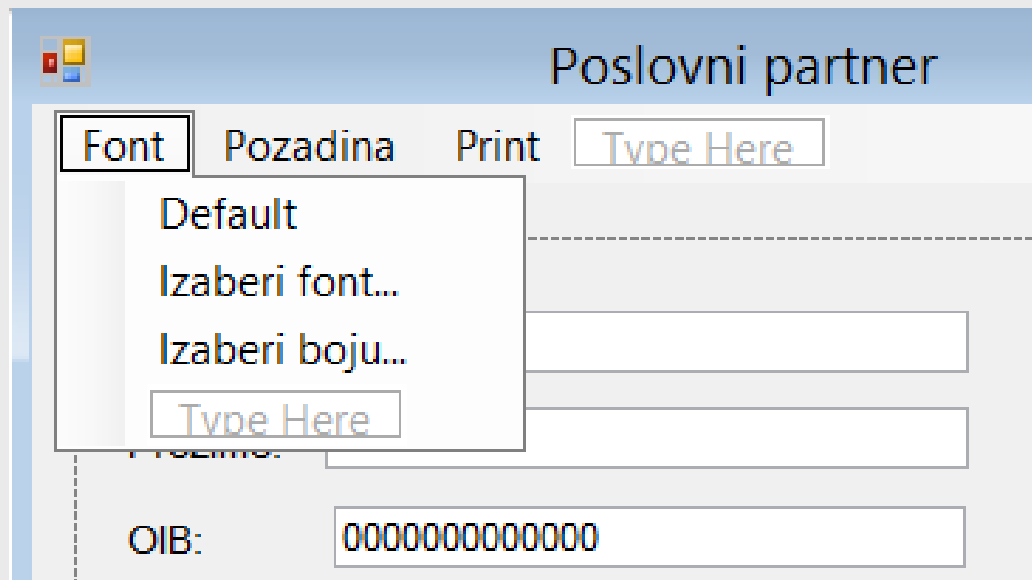
Primjer izbornika

❑ Primjer: GUNPartner

- izgled izbornika u dizajnu
- korijen sustava izbornika (MenuStrip) i stavke (ToolStripMenuItem)
- desni klik + Edit items

Događaji izbornika

- Primjer, klik mišem na stavku izaberi font...



```
System.Windows.Forms.MenuStrip menuStrip;
...
this.MainMenuStrip = this.menuStrip;
...
private void izaberiFontToolStripMenuItem_Click(
    object sender, EventArgs e)
{
    //...
}
```

Dijalozi

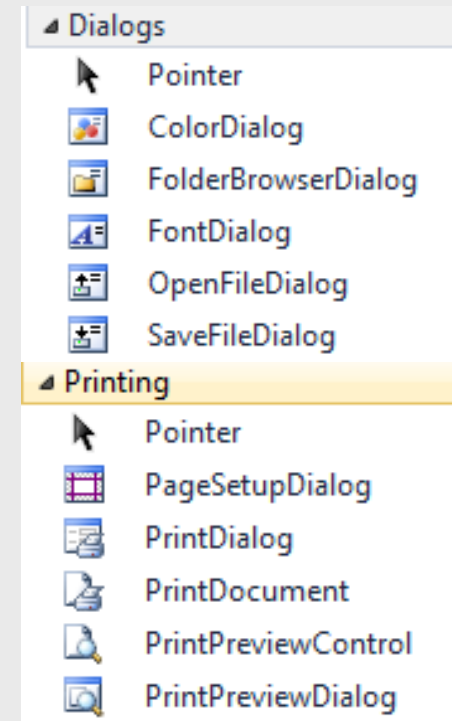
- ❑ **Dijalog** – posebna, predefinirana vrsta forme za jednostavnu interakciju i odabir vrijednosti korisnika u specifičnim situacijama

- ❑ **Vrste dijaloga, npr.**

- promjena tipa pisma (*FontDialog*), boje slova,
- promjena boje pozadine (*ColorDialog*)
- tisak (*PrintDialog* i *PrintDocument*)
- dijalog za obradu datoteka (*OpenFileDialog*)

- ❑ **Svi ugrađeni dijalozi imaju postupke**

- `ShowDialog` – prikaz dijaloga
 - vraća `DialogResult` kao rezultat što se dogodilo (enumeracija `None`, `OK`, `Cancel`, `Abort`, `Retry`, `Ignore`, `Yes`, `No`)
- `Dispose` – oslobađa dijalogom zauzete resurse



Predefinirani dijalozi

❑ Predefinirani dijalozi i najvažnija svojstva (pogledati Help)

- OpenFileDialog - CheckFileExists, FileName, Filter, InitialDirectory, ShowReadOnly
- SaveFileDialog - CheckFileExists, FileName, Filter, InitialDirectory, OverwritePrompt
- FolderBrowserDialog - SelectedPath, ShowNewFolderButton
- ColorDialog - Color, AnyColor, FullOpen
- FontDialog - Font, FontMustExist
- PageSetupDialog - Document (*PrintDocument*), PageSettings
- PrintDialog - Document (*PrintDocument*), PrinterSettings
- PrintPreviewDialog - Document (*PrintDocument*)

❑ Nalaze se u ***System.Windows.Forms***

Primjeri dijaloга

❑ OpenFileDialog – odabir slike

```
if (openFileDialogSlika.ShowDialog() != DialogResult.Cancel) {  
    string NazivSlike = openFileDialogSlika.FileName;  
}
```

❑ FontDialog – odabir tipa pisma

- `this.Font = fontDialog.Font;`

❑ PrintDialog – odabir opcija za zapisivanje

- `(System.Drawing.Printing.)PrintDocument`
 - dokument za zapisivanje
- `System.Drawing.Printing.PrintPageEventArgs.Graphics`
 - Dohvaća grafiku za iscrtavanje stranice (dokumenta) za zapisivanje

```
printDocument.PrinterSettings = printDialog.PrinterSettings;  
printDocument.Print();  
...  
e.Graphics.DrawString(tekstZaPrintanje, this.Font, Brushes.Black,  
    new PointF(180, 50));
```

Poruke i resursi

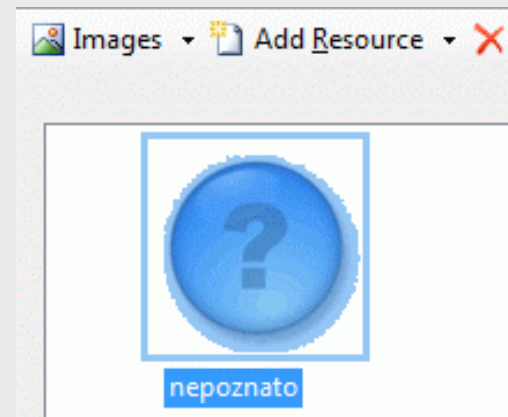
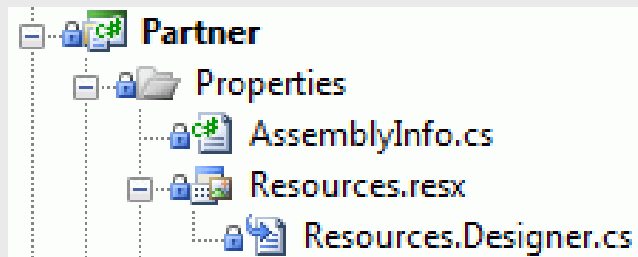
❑ Razred MessageBox

- Postupak **Show** – prikaz poruke

```
private void buttonSpremi_Click(object sender, EventArgs e)
{
    MessageBox.Show("Spremljeno!");
}
```

❑ Resources

- spremište stringova, datoteka, slika, ...
- *Form1.resx* – spremište resursa forme *Form1*
- Properties projekta \ *Resources.resx*
 - spremište koje upotrebljava cijela aplikacija



- Dohvat objekata iz spremišta *Resources.resx* :
[nazivImenika].Properties.Resources.nazivElementa

```
pictureBoxSlika.Image = Properties.Resources.nepoznato;
```


MessageBox

❑ Javni razred MessageBox

- Postupak Show – prikaz poruke, više oblika, najčešći su:

- `DialogResult Show(string text);`
- `DialogResult Show(string text, string caption);`
- `DialogResult Show(string, string, MessageBoxButtons);`
- `DialogResult Show(string, string, MessageBoxButtons, MessageBoxIcon);`





❑ MessageBoxButton i MessageBoxIcon

- `MessageBoxButtons.OK`
- `MessageBoxButtons.OKCancel`
- `MessageBoxButtons.YesNo`
- `MessageBoxButtons.YesNoCancel`
- `MessageBoxButtons.RetryCancel`
- `MessageBoxButtons.AbortRetryIgnore`

❑ enum DialogResult

- `Abort, Cancel, Ignore, No, None, OK, Retry, Yes`

❑ Primjer: buttonSpremi_Click

MessageBox Icons	Icon
<code>MessageBoxIcon.Exclamation</code>	
<code>MessageBoxIcon.Information</code>	
<code>MessageBoxIcon.Question</code>	
<code>MessageBoxIcon.Error</code>	

Validacija unosa podataka

❑ Događaji kontrole

- `Validating (object sender, CancelEventArgs e)` – okida provjeru ispravnosti podataka u trenutku kad kontrola treba biti napuštena
 - postavljanje `e.Cancel = false` omogućuje napuštanje kontrole
 - postavljanje `e.Cancel = true` blokira napuštanje kontrole u slučaju pogrešnog unosa (ako na formi nije mijenjano svojstvo `AutoValidate`)
 - problem: ako postoje neispravni podaci, nije moguće poduzeti nikakvu akciju sve dok se pogreška ne popravi pa čak ni zatvoriti formu
 - rješenje: u `(Form1_FormClosing` postaviti `e.Cancel = false`
- `Validated` – nakon što je provjera obavljena

❑ Svojstva kontrole

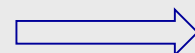
- `CausesValidation` – `false`: ne aktivira `Validating` i `Validated`

❑ Forma

- `this.AutoValidate` – o(ne)mogućuje implicitnu validaciju
- `this.ValidateChildren()` – pokreće `Validating` i `Validated` za kontrole

❑ Komponenta `ErrorProvider` – poruka o pogrešnom unosu

- ikona uskličnika se prikazuje dok unos nije valjan
- `SetError(Control, string)` – postavlja poruku o pogrešci uz kontrolu




Primjer validacije pri unosu u zaslonsku masku

❑ Primjer: 📁 GUNPartner

- želimo upozoriti na unos neispravnih podataka
 - Ime i prezime moraju biti uneseni i ne smiju sadržavati brojeve
 - OIB mora biti niz brojeva duljine 11 znakova
 - ...

Font Pozadina Print

Ime: 

Prezime:

OIB:

Ime/prezime treba biti valjanog fomata.

ErrorProvider

❑ **ErrorProvider** – prikaz informacije o pogrešci

❑ **Svojstva**

- `BlinkRate` – učestalost treptanja
- `BlinkStyle` – `BlinkIfDifferentError`, `AlwaysBlink`, `NeverBlink`

❑ **Postupci**

- `SetError(Control, string)` – kontrola i objašnjenje

❑ **Prikaz poruke iz postupka za validaciju, npr. `ValidacijaNaziv`**

- provjera valjanosti formata za ime/prezime
- postavljanje poruke za `errorProvider`

```
private bool ValidacijaNaziv(TextBox textBox){  
    if (textBox.Text == ""){  
        errorProvider.SetError(textBox, "Unesite ime/prezime.");  
        return false;  
    }  
    ...  
}
```

❑ **Ili automatski postavljanjem svojstva *DataSource* na podatak koji implementira *IDataErrorInfo***

Validacija kontrole i validacija forme

❑ Implementacija događaja *Validating* na kontroli `textBoxIme`

- `e.Cancel` omogućava/onemogućava napuštanje kontrole (ako svojstvo forme *AutoValidate* nije na *EnableAllowFocusChange* ili *Disable*)

```
private void textBoxIme_Validating(object sender, CancelEventArgs e)
{
    if (ValidacijaNaziv((TextBox)sender)) {
        e.Cancel = false; // validacija ok, dozvoliti napuštanje kontrole
    }
    else {
        e.Cancel = true; // validacija neuspješna, forsirati popravak
    }
}
```

❑ Klikom na gumb *Spremi* pozivamo postupak za validaciju

- ako su svi podaci ispravni, `this.ValidateChildren()` vraća *true*

```
if (this.ValidateChildren()) {
    MessageBox.Show("Spremljeno!");
    OcistiFormu();
}
```

ToolTip

❑ **ToolTip – podsjetnik kontrole kojoj je ToolTip pridružen**

■ Svojstva

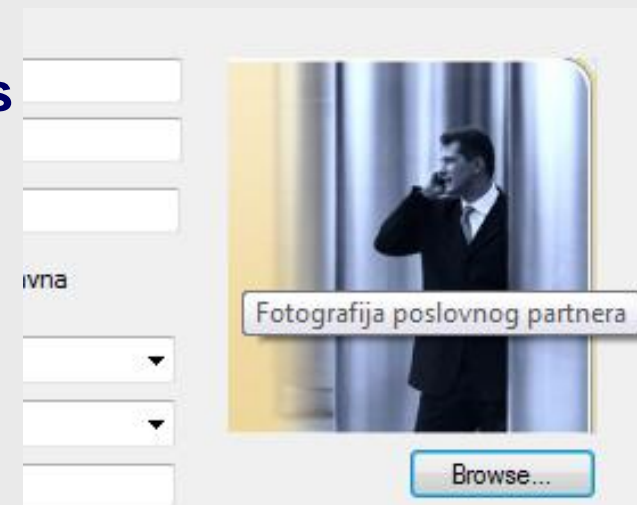
- `AutomaticDelay` – vrijednost za automatsku postavku ostalih vremena
- `InitialDelay` – vrijeme do prvog prikaza (= `AutomaticDelay`)
- `AutoPopDelay` – vrijeme trajanja prikaza (= `AutomaticDelay*10`)
- `ReshowDelay` – vrijeme do ponovnog prikaza (= `AutomaticDelay/5`)
- `ShowAlways` – aktivan i kad kontrola kojoj je pridružen nije u fokusu

■ Postupci (uobičajeno u forma.Designer.cs)

- `SetToolTip(Control, string)` – kontroli postavlja objašnjenje, u dizajnu se vidi u popisu svojstava kao "ToolTip on control"
- `SetToolTip(Control)` – vraća objašnjenje kontrole u pogonu

❑ **Primjer: GUI \ Partner \ Form1.Designer.cs**

```
ToolTip toolTip = new  
    ToolTip(this.components);  
...  
toolTip.SetToolTip(pictureBoxSlika,  
    "Fotografija poslovnog partnera");
```



Dinamičko kreiranje kontrola

❑ Primjer GUN\KreiranjeKontrola

■ Kreiranje gumba klikom na formu

```
private void Form1_Click(object sender, EventArgs e)
{
    Button b = new Button();
    b.Text = "Gumb " + this.Controls.Count;
    Point mousePoint = PointToClient(MousePosition);
    b.Location = new Point(mousePoint.X, mousePoint.Y);

    b.Click += new System.EventHandler(this.ButtonClick);
    this.Controls.Add(b);
}

private void ButtonClick(object sender, System.EventArgs e)
{
    MessageBox.Show(sender.ToString());
}
```

Prijenos parametara među formama

- ❑ Forma poziva drugu formu i prosljeđuje joj parametre
- ❑ Izmjene na drugoj formi koriste se za ažuriranje na pozivajućoj formi

Unos podataka o partneru

Ime: Ivo

Prezime: Ivić

OIB: 12345678901

Tip: ☒ Fizička ☐ Pravna

Adresa

Država:

Mjesto:

Ulica i broj:

Unesi adresu ...

Spremi

Unos adrese

Država: Hrvatska

Mjesto: Split

Ulica i broj: Vukovarska 15

Spremi Odustani

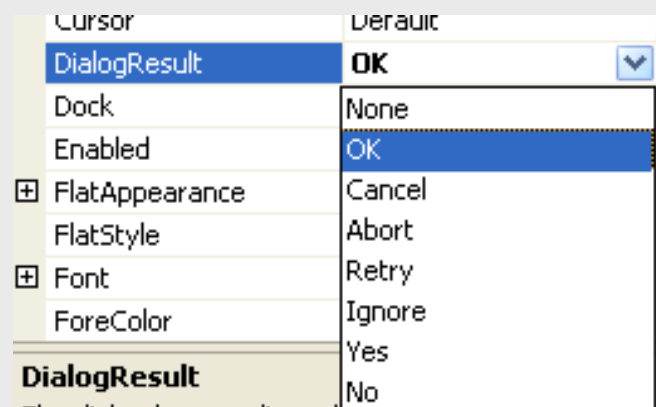
Prijenos parametara među formama (2)

❑ Primjer: GUI \ PrijenosParametara

- Prijenos parametara iz pozvane forme u pozivajuću – FormaAdresa
- FormaAdresa ima svojstva Drzava, Mjesto i Ulica koje su omotač oko pripadnih kontrola za unos

```
public string Drzava {  
    get {  
        if (comboBoxDrzava.SelectedItem == null) return null;  
        else return comboBoxDrzava.SelectedItem.ToString();  
    }  
    set { comboBoxDrzava.SelectedItem = value; }  
}
```

- Gumbi *Spremi* i *Odustani* imaju postavljena svojstva DialogResult na *OK*, odnosno *Cancel*



Prijenos parametara među formama (3)

❑ Prijenos parametara iz pozvane forme u pozivajuću – FormaUnos

- FormaUnos instancira formu FormaAdresa i otvara je kao dijalog

```
private void buttonUnos_Click(object sender, EventArgs e) {  
    FormAdresa f = new FormAdresa();  
    ...  
    if (f.ShowDialog() == DialogResult.OK) {  
        textBoxDrzava.Text = f.Drzava;  
        textBoxMjesto.Text = f.Mjesto;  
        textBoxAdresa.Text = f.Ulica;  
    }  
}
```

- Ako je na formi FormaAdresa odabran gumb *Spremi*, FormaUnos dohvaća vrijednosti Drzava, Mjesto i Ulica forme FormaAdresa i postavlja ih kao vrijednosti svojim pripadajućim kontrolama
- Ako je odabran gumb *Odustani* vrijednosti se ne dohvaćaju
- Svojstvo *DialogResult* postavljeno na *Cancel* na gumbu *Odustani*

Prijenos parametara među formama (4)

❑ Prijenos parametara iz pozivajuće forme u pozvanu – FormaUnos

- Jedan način, slanje parametara u konstruktor pozvane forme
- `FormaAdresa f = new FormaAdresa(textBoxDrzava.Text);`
- Drugi način, promjenom javnih svojstava forme, kao u našem primjeru
- Referencom na formu `FormaAdresa`

```
private void buttonUnos_Click(object sender, EventArgs e) {  
    FormaAdresa f = new FormaAdresa();  
    f.Drzava = textBoxDrzava.Text;  
    f.Mjesto = textBoxMjesto.Text;  
    f.Ulica = textBoxAdresa.Text;  
  
    if (f.ShowDialog() == DialogResult.OK) { ... }  
}
```

- Rješenje je napravljeno po uzoru na dijaloge (npr. *OpenFileDialog*)

Nasljeđivanje formi

❑ Jednostavno nasljeđivanje formi

```
public partial class Form2 : Partner.Form1
{
    public Form2()
    {
        InitializeComponent();
    }
    private void InitializeComponent()
    {
        this.Text = this.Text + " naslijeđena";
        this.BackColor = System.Drawing.Color.Yellow;
    }
}
```

❑ Nasljeđivanje u dizajnu

- Izvedena forma dodaje se s *Project* → *Add* → *New Item* → *WindowsForms* → *InheritedForm*
- Mogu se postavljati svojstva izvedenoj formi ali ne njezinim kontrolama (sve dok je svojstvo *Modifiers* na *Private*)

Vlastite kontrole

❑ Vlastite kontrole (*custom controls*)

- Elementi sučelja koje stvara korisnik razvojnog pomagala – programer
- Nakon toga može ih se koristiti u različitim programima, kao i druge preddefinirane kontrole, odabirom iz kutije s alatima

❑ Tipovi vlastitih kontrola:

- *User kontrole ("korisničke")*
 - Nasljeđivanje `System.Windows.Forms.UserControl`
 - Kombinacija više postojećih kontrola u logičku cjelinu
- *Owner-draw kontrole ("nacrtane")*
 - Nasljeđivanje `System.Windows.Forms.Control`
 - Crtanje kontrole ispočetka
- Naslijeđene kontrole
 - Nasljeđivanje kontrole koja je najsličnija onoj koju želimo napraviti
 - Dodavanje novih svojstava i postupaka

Primjer vlastite "korisničke" kontrole

❑ **Windows Forms Control Library projekt**

- rezultat je .dll datoteka (*class library*)

❑ **Kontrola nasljeđuje razred UserControl**

- `public partial class VlastitiTextBox : UserControl`

❑ **Primjer GUI\VlastiteKontrole – razred VlastitiTextBox**

- Implementiramo funkcionalnosti koje želimo, npr. promjena boje prilikom unosa teksta (obrada događaja *Enter*)



```
private void textBox_Enter(object sender, EventArgs e) {  
    oldText = this.textBox.Text;  
    oldColor = this.BackColor;  
    this.BackColor = Color.DeepSkyBlue;  
}
```

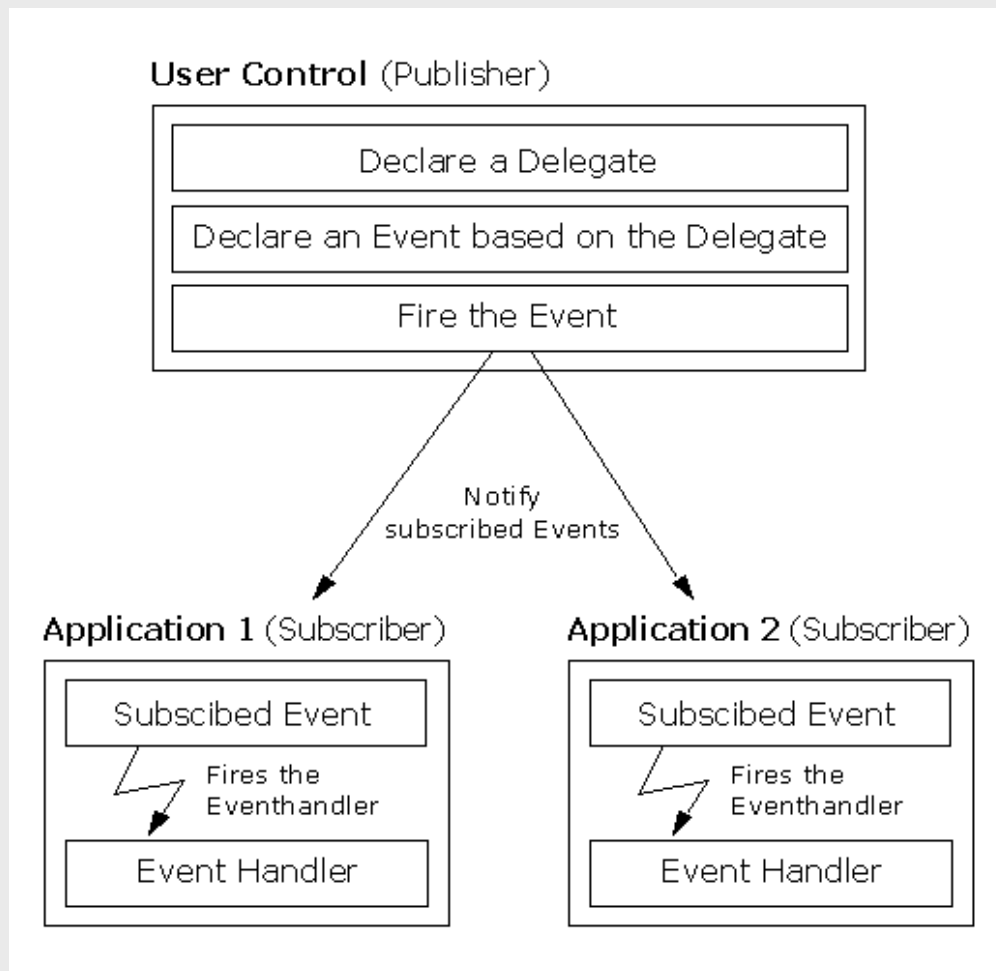
- Implementiramo svojstva vlastite kontrole kojima pristupamo iz aplikacije

```
public string Labela {  
    get { return labelText.Text; }  
    set { labelText.Text = value; }  
}
```

Događaji na vlastitoj kontroli

❑ Model izdavač-pretplatnik (publisher-subscriber)

- Objekt “objavljuje” događaj, a aplikacija se na njega “pretplaćuje”
- U ovom modelu vlastita kontrola je izdavač



Ugradnja “vlastitog” događaja

❑ Naša kontrola ima događaj promjene sadržaja

- definiran delegatom koji određuje tip obrade

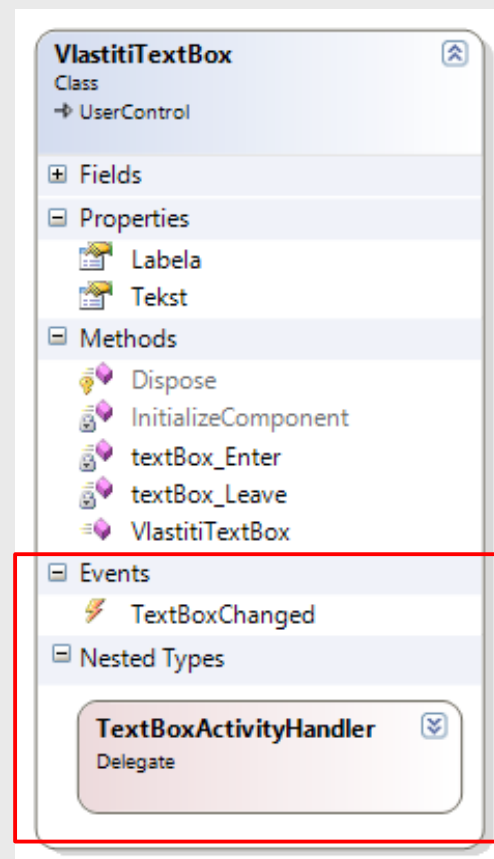
```
public delegate void TextBoxActivityHandler(object sender, EventArgs e);  
public event TextBoxActivityHandler TextBoxChanged;
```

- Interno se stvara novi razred `TextBoxActivityHandler` koji nasljeđuje razred `MultiCastDelegate`


❑ Pokretanje metode za obradu događaja

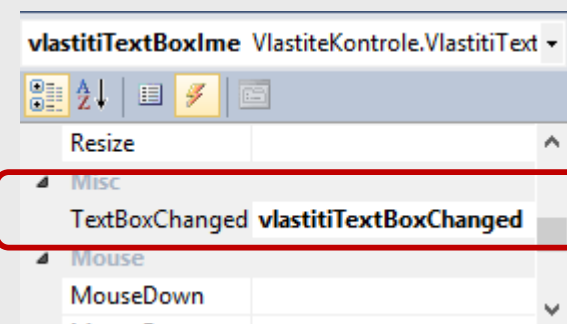
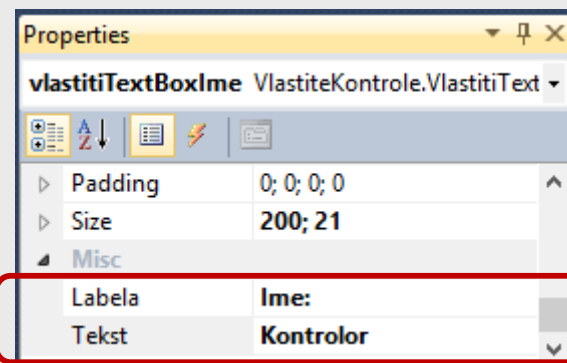
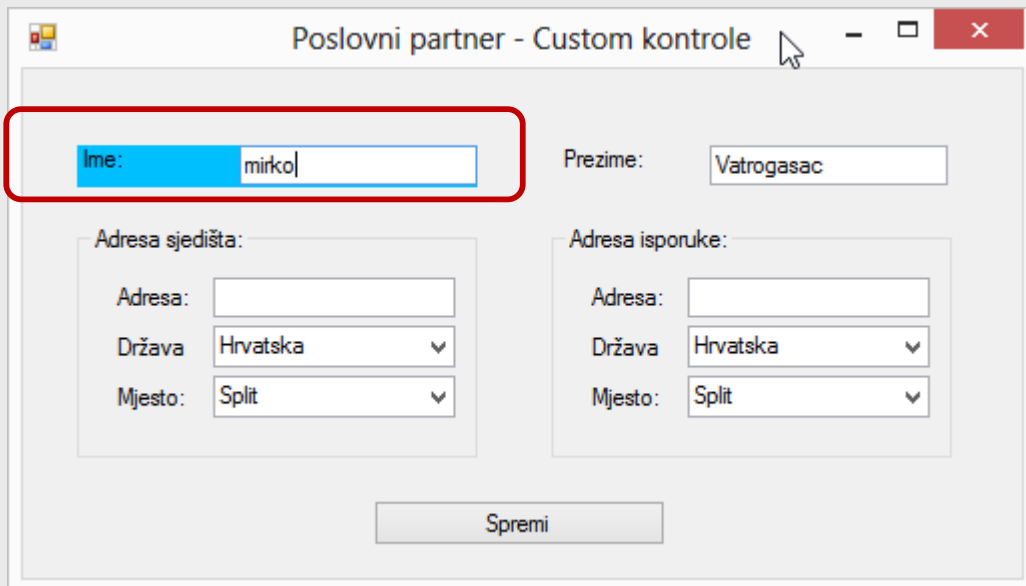
- obavlja se pozivom pretplatnika na događaj
- pretplatnika definira aplikacija koja koristi kontrolu
 - tamo to bude metoda za obradu događaja
- kreator kontrole određuje kada će događaj okinuti
 - u našem primjeru po napuštanju textbox-a

```
private void textBox_Leave(object sender, ...) {  
    ...  
    // ako postoji pretplatnik na događaj  
    if (TextBoxChanged != null) {  
        // pozovi pretplatnika  
        TextBoxChanged(this, e);  
    }  
}
```



Upotreba vlastite kontrole

- ❑ Dodavanje vlastite kontrole (VlastiteKontrole.dll) u projekt
- ❑ Primjer  GUI\Custom
 - Desni klik na *Toolbox* → *Choose Items* → *Browse...* *VlastiteKontrole.dll*
 - Ili, desni klik na projekt – *Add reference* - *Project* - *VlastiteKontrole*
- ❑ Dodamo kontrolu s *ToolBoxa* i postavljamo joj svojstva
 - Na popisu su svojstva i događaji koje smo sami definirali



Obrada događaja vlastite kontrole

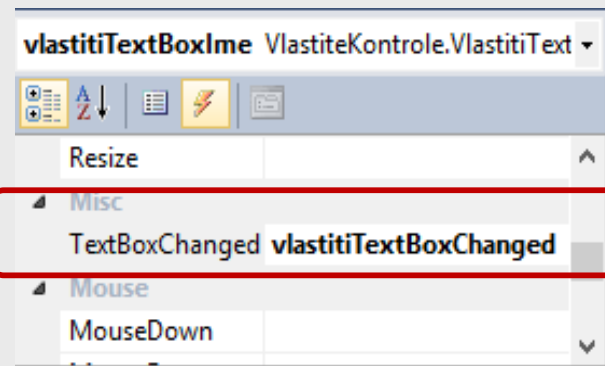
❑ Implementiramo rukovatelj događajem

```
private void vlastitiTextBox_Changed(object sender, EventArgs e)
{
    VlastitiTextBox t = (VlastitiTextBox)sender;
    t.Tekst = t.Tekst.Substring(0, 1).ToUpper()
        + t.Tekst.Substring(1).ToLower();
}
```

❑ te ga povežemo s instancom vlastite kontrole

```
this.vlastitiTextBoxIme.TextBoxChanged += new
    VlastiteKontrole.VlastitiTextBox.TextBoxActivityHandler
    (this.vlastitiTextBox_Changed);
```

- povezivanje (pretplatu)
radimo kao za sve druge kontrole



Višenitnost i paralelno programiranje u C#-u

❑ Nit ili dretva (eng. *thread*)

- Osnovne jedinice u kojima OS raspodjeljuje procesorsko vrijeme
- Višenitnost (*multithreading*) => Više niti može se izvoditi unutar jednog procesa. Višenitni program simulira istovremeno izvođenje različitih niti

❑ C# i višenitnost

- Asinkrono pozivanje postupaka
 - delegat ima mogućnost asinkronog poziva (*BeginInvoke*, *EndInvoke*)
- Kontrola *BackgroundWorker*
- Stavljanje postupka u postojeći red niti (razred *ThreadPool*)
- Stvaranje nove niti (razred *Thread*)
- TPL (eng. *Task Parallel Library*)
 - Stvaranje jednog ili više zadataka (razred *Task*)
 - Paralelno izvršavanje dijelova koda (razred *Parallel*)
 - Paralelni LINQ upiti (PLINQ)
- C# 5.0 / .NET 4.5: korištenje asinkronih verzija postupaka (ako takvi postoje za određenu namjenu) + ključne riječi *async* i *await*

Primjer potrebe za paralelnim izvršavanjem

❑ Izračun broja π i istovremeni prikaza statusa - GUI \ RacunanjePi

- Za provjeru prekida postupka koristi se zastavica (boolean varijabla *abort*)
 - Može se koristiti struktura *CancellationToken*
- Podizanje vlastitog događaja nakon svakog koraka izračuna

```
public delegate void ShowProgressDelegate(string pi, int totalDigits,
int digitsSoFar);
```

```
public class PiCalculator {
    public event ShowProgressDelegate ProgressChanged;
    ...
    public string CalcPi(int digits) {
        abort = false;
        StringBuilder pi = new StringBuilder("3.", digits + 2);
        for (int i = 0; i < digits && !abort; i += 9) {
            ...
            //prikaz trenutnog stanja podizanjem vlastitog događaja
            ProgressChanged(pi.ToString(), digits, i + digitCount);
            ...
        }
        return pi.ToString();
    }
    ...
}
```

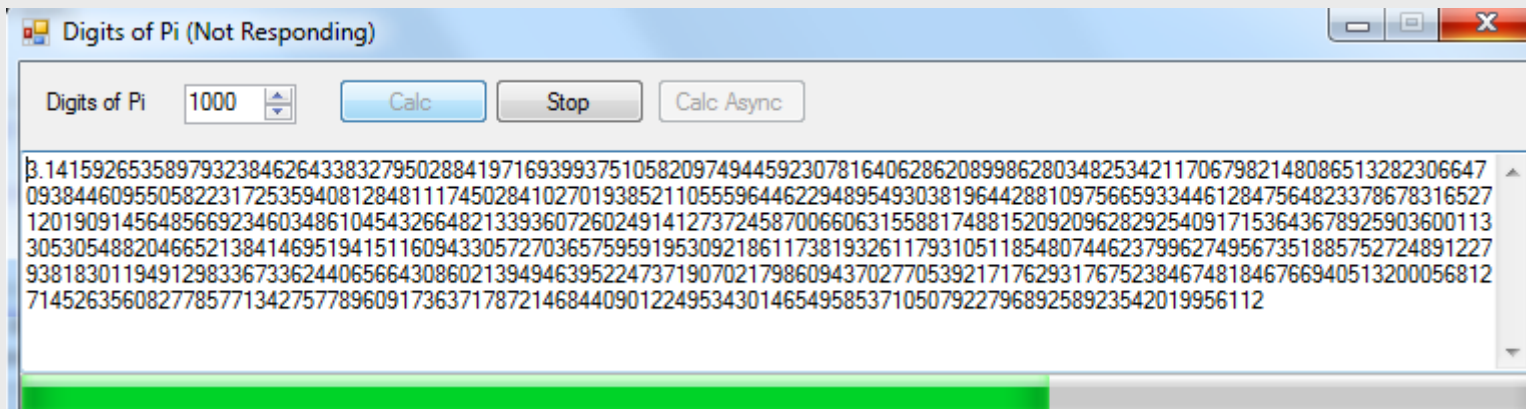
Problem jednonitnog izvršavanja

❑ Primjer GUI \ RacunanjePi \ Form1.cs

- Instancira se postupak za izračun broja pi i pretplaćuje se na događaj promjene status
- Izračun započinje na istoj niti

```
void btnCalc_Click(object sender, EventArgs e) {  
    ...  
    int digitsToCalc = (int)digitsUpDown.Value;  
    piCalculator = new PiCalculator();  
    piCalculator.ProgressChanged += ShowProgress;  
    string pi = piCalculator.CalcPi(digitsToCalc);  
}
```

- Problem: Forma ne reagira dok postupak izračuna ne završi



Stvaranje zadatka u pozadini


❑ Primjer  GUI \ RacunanjePi \ PiCalculator.cs

❑ Koriste se razredi *Task* i/ili *Task<TResult>* (alternativno *Task.Run*)

- Za definiranje postupka u pozadini bez povratne vrijednosti (ili tipa *TResult*)
- Konstruktor razreda *Task* očekuje
 - Delegat tipa *Action* ili *Action<object>* : *void* postupak bez argumenata ili s jednim argumentom tipa *object*
- Konstruktor razreda *Task<TResult>* očekuje
 - Delegat tipa *Func<TResult>* ili *Func<object, TResult>* : postupak bez argumenata (ili s argumentom tipa *object*) koji vraća objekt tipa *TResult*
- Ostali (opcionalni) parametri konstruktora uključuju argument koji se proslijeđuje postupku te razred za signalizaciju otkazivanja zadatka
- Umjesto eksplicitno definiranih delegata uobičajeno je koristiti lambda izraze

```
public Task<string> CalcPiAsync(int digitsToCalc){  
    var piTask = new Task<string>(() => { //lambda izraz  
        string pi = CalcPi(digitsToCalc);  
        return pi;  
    });  
    piTask.Start();  
    return piTask;
```

Čekanje zadatka i asinkroni postupci

- ❑ **Način korištenja *await* + postupak koji vraća Task ili Task<TResult>**
 - Pozivatelj čeka na dovršetak zadatka, ali se istovremeno omogućava formi da reagira na druge događaje
 - Kôd iza *await* nastavlja se izvršavati tek nakon dovršetka zadatka
 - *await* + zadatak tipa Task<TResult> vraća rezultat tipa TResult
 - Postupak u kojem se koristi *await* mora se označiti s *async*
 - *Napomena: Povratna vrijednost iz postupka oblika async Task<TResult> je tipa TResult.*
 - Rukovatelji događaja mogu se označiti s *async*
- ❑ **Primjer  GUI \ RacunanjePi \ Form1.cs**
 - Izračun pokrenut kao zadatak (*Task*) (vidi prethodni slajd)

```
private async void btnCalcAsync_Click(...) {  
    ...  
    piCalculator = new PiCalculator();  
    piCalculator.ProgressChanged += ShowProgress;  
    string pi = await piCalculator.CalcPiAsync(digitsToCalc);  
    ...  
}
```

ProgressBar

❑ Kontrola za informiranje o napredovanju nekog procesa

❑ Svojstva

- Minimum, Maximum – najmanja i najveća vrijednost
- Value – trenutni položaj oznake
- Step – korak promjene položaja oznake pri pozivu PerformStep postupka

❑ Postupci

- Increment – promjena vrijednosti za navedenu veličinu
- PerformStep – promjena vrijednosti za veličinu Step

❑ Primjer GUI\RacunanjePi

```
void ShowProgress(string pi, int totalDigits, int digitsSoFar)
{
    if (this.InvokeRequired) { ... }
    else{
        piTextBox.Text = pi;
        piProgressBar.Maximum = totalDigits;
        piProgressBar.Value = digitsSoFar;
    }
}
```


Pristup kontrolama korisničkog sučelja iz drugih niti

❑ Primjer GUI \ RacunanjePi \ Form1.cs

- Promjena vrijednosti ili stanja kontrole u grafičkom sučelju treba biti izvršena iz niti koja je stvorila tu kontrolu
- Svojstvo *InvokeRequired* vraća odgovor da li je pozivatelj na drugoj niti

```
public delegate void ShowProgressDelegate(string pi, int
                                         totalDigits, int digitsSoFar);

void ShowProgress(string pi, int totalDigits, int digitsSoFar){
    if (this.InvokeRequired){ //poziv izvan primarne niti?
        ShowProgressDelegate showProgress = ShowProgress;
        this.BeginInvoke(showProgress, pi, totalDigits,
                        digitsSoFar);
    }
    else{
        piTextBox.Text = pi;
        piProgressBar.Maximum = totalDigits;
        piProgressBar.Value = digitsSoFar;
    }
}
```

Dodatni primjeri

❑ RPPP04-dodatak.pdf

- još neki primjeri kontrola
- primjer računanja broja π izveden na više načina ostvarivanja višenitnosti