


# **Grafičko korisničko sučelje**

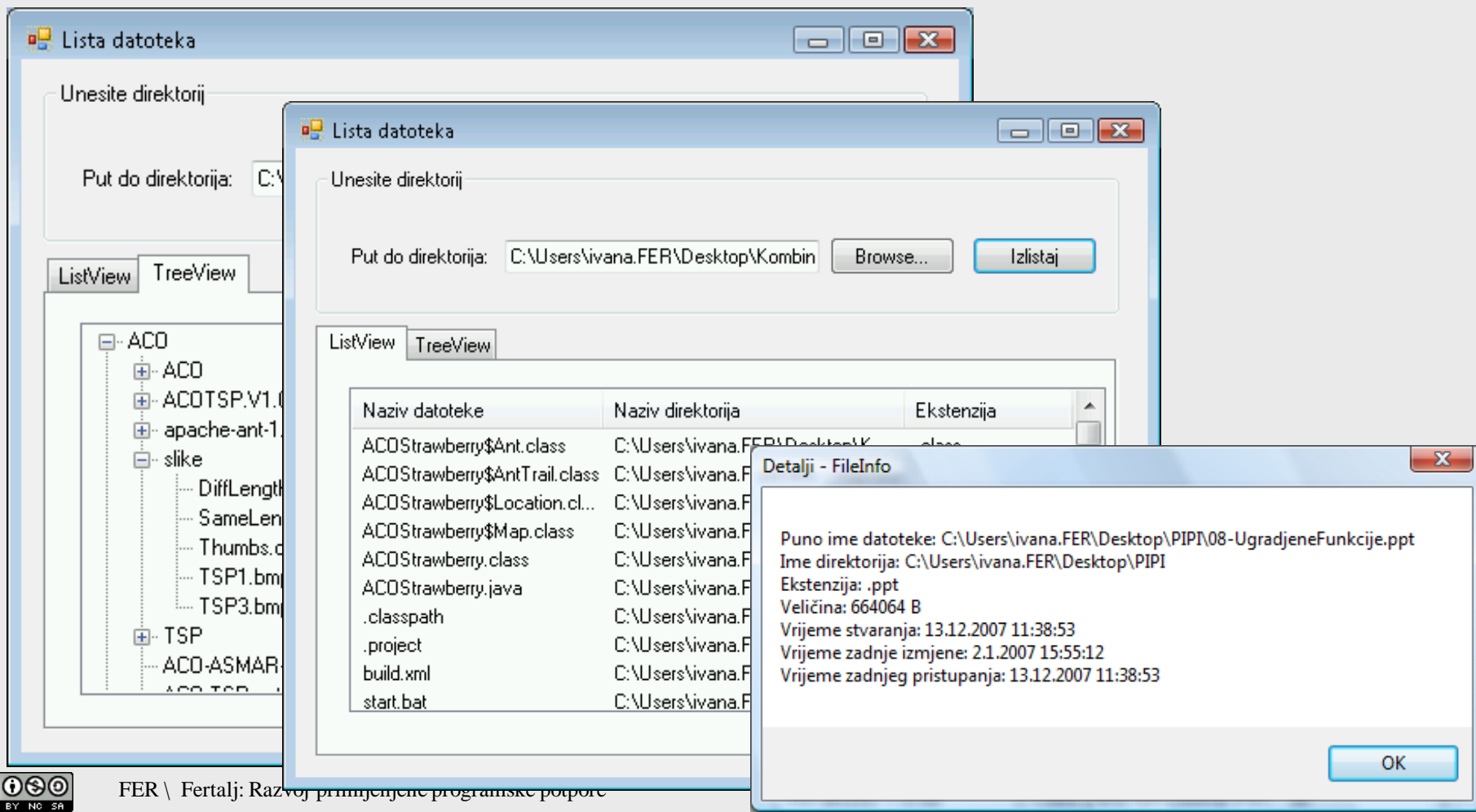
## **Fakultativne folije**

---

**2014/15.04 - dodatak**

# Primjer kontrola *TabControl*, *ListView*, *TreeView*

- ❑ Primjer:  **GUIListaDatoteka** - prikaz datoteka/direktorija
  - Desni klik na element u *ListView* – detalji o odabranom elementu
  - *ListView* svojstvo *ContextMenuStrip* i događaj *MouseDown*
  - Kontrola *contextMenuStrip* i događaj *detaljiToolStripMenuItem\_Click*



# TabControl

## ❑ **TabControl - Kontrola s karticama (tabovima)**

### ■ Svojstva

- Alignment – poravnanje tabova (Top, Bottom, Left, Right)
- Appearance – izgled tabova (Normal, Buttons, FlatButtons)
- HotTrack – dinamičko isticanje (highlight) taba pri prolasku miša
- ImageList – kolekcija bitmapa za tabove
- ItemSize – veličina tabova
- Multiline – tabovi u više redaka pri sužavanju kontrole
- SizeMode – automatsko podešavanje širine tabova (Normal, Fixed, FilledToRight)
- **TabPage** – kolekcija tabova
- **TabCount** – broj tabova
- **SelectedIndex** – indeks odabranog taba
- **SelectedTab** – odabrani tab

### ■ Događaji

- **SelectedIndexChanged** – promjena svojstva SelectedIndex

## ❑ **TabPage**

### ■ Svojstva # otprije poznata

- AutoScroll, BackgroundImage, ImageIndex, Text

# ListView

## ❑ Prikaz kolekcije elemenata u mreži

### ■ Svojstva

- View – enum { Details, LargeIcon, List, SmallIcon }
- SmallImageList, LargeImageList – liste slika za *SmallIcon* i *LargeIcon*
- AllowColumnReorder, AutoArrange, GridLines – prilagodba prikaza
- CheckBoxes – *CheckBox* elementi
- FullRowSelect – označava se cijeli redak odabranog elementa
- Multiselect – odabir više elemenata
- Columns – lista zaglavlja stupaca,
  - ColumnHeader razred sa svojstvima: Index, Text, Width
- Items – lista ListViewItem elemenata
  - Svojstva: Count,
  - Postupci: Add, Clear, Insert, Remove
- CheckedItems, SelectedItems – lista označenih, lista odabranih

### ■ Događaji

- SelectedIndexChanged, StyleChanged, ...

## ❑ ListViewItem element

### ■ Svojstva:

- Index – indeks elementa u listi
- ImageIndex – indeks slike u SmallImageList, odnosno LargeImageList
- Selected – element je označen

# TreeView

## ❑ **TreeView – stablasta lista čvorova**

### ■ Svojstva

- CheckBoxes – *CheckBox* čvorovi
- Nodes – *TreeNodeCollection*
- ShowLines – prikaz poveznica između čvorova
- ShowPlusMinus – prikaz znaka za širenje/skupljanje čvora
- TopNode – čvor na vrhu

### ■ Postupci

- CollapseAll, ExpandAll – širenje i skupljanje prikaza

### ■ Događaji

- BeforeSelect, AfterSelect
- BeforeCollapse, AfterCollapse
- BeforeExpand, AfterExpand

## ❑ **Nodes (TreeNodeCollection) - kolekcija čvorova**

### ■ Postupci

- Add, Remove ...

## ❑ **TreeNode – čvor**

### ■ Svojstva

- Text
- FullPath – put do čvora
- Index – indeks čvora
- Nodes – kolekcija djece
- FirstNode, LastNode - djeca
- NextNode, PrevNode – braća
- IsExpanded, IsSelected

### ■ Postupci

- Collapse, Expand – širenje i skupljanje prikaza odabranog

# *ImageList*

## ❑ Razred s listom *Image* objekata

- `Images` – lista *Image* objekata
  - Svojstva: `Count`, `Empty` – svojstva liste
  - Postupci: `Add`, `Clear`, `Remove` – rukovanje elementima liste
- `ImageSize` – veličina bitmape pojedinog *Image* objekta u listi
  - standardno (16;16), najviše (255;255)

## ❑ **Button i neke druge kontrole imaju svojstva**

- `ImageList` referenca
- `ImageIndex` – indeks *ImageList* člana

## ❑ Definiranje bitmapa u dizajnu:

- `imageList1.Images` (Properties ... collection)

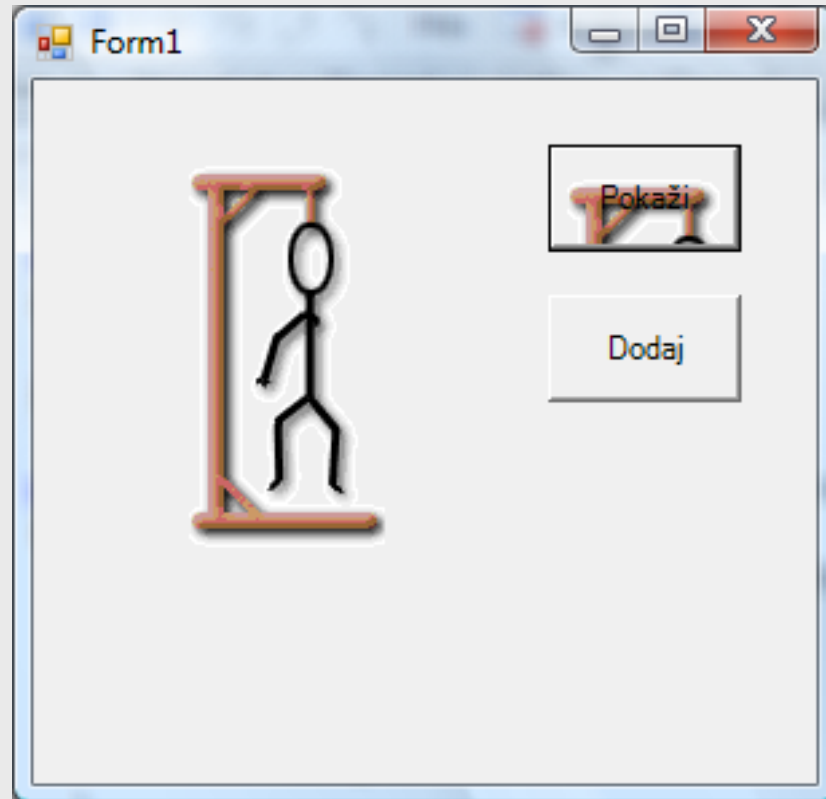
## ❑ Dodavanje bitmape programski:

- `imageList1.Images.Add(Image.FromFile(openFileDialog1.FileName));`

# Zadaci za vježbu

## ❑ Napraviti formu za prikaz vješala za igru Vješala.

- Unaprijed pripremljen niz slika prikazati u *ImageList*
  - Slike su pohranjene u binarnom formatu u Form1.resx
- Klikom na gumb Prikaži dodaje se sljedeća slika
- Klikom na gumb Dodaj se može dodati nova slika
- Primjer:
  - GUI \ Dodatak \ Vjesala



# ***RichTextBox***

## ❑ **Jača verzija TextBox-a (mini word procesor)**


## ❑ **Svojstva i postupci**

- `RichTextBoxStreamType` – vrsta teksta
  - `PlainText`, `RichText`, `UnicodePlainText`, itd.
- `LoadFile` - Čitanje dokumenta
- `SaveFile` - Pisanje dokumenta
- `SelectedText`, `SelectedRTF` – označeni tekst
- `SelectionLength` – duljina označenog teksta
- `SelectionFont` – Font označenog teksta
- `SelectionColor` – Boja označenog teksta

## ❑ **Primjer: GUI \ Dodatak \ RichTextBoxPad**



# ScrollBar

- ❑ Klizne trake, pomične trake
- ❑ *HScrollBar* – vodoravna klizna traka
- ❑ *VScrollBar* – okomita klizna traka
- ❑ Svojstva:
  - `Minimum` – najmanja vrijednost kliznika kontrole
  - `Maximum` – najveća vrijednost kliznika kontrole
  - `SmallChange` – korak vrijednosti za male pomake kliznika ili klik na rub
  - `LargeChange` – vrijednost inkrementa kad korisnik klikne unutar kontrole
  - `Value` – vrijednost kontrole, predstavlja položaj kliznika
- ❑ Primjer:  **GUI \ Dodatak \ ScrollBars**

# TrackBar

## ❑ Tračna traka, "Tračnica"

## ❑ Svojstva:

- `Minimum` – najmanja vrijednost kliznika kontrole
- `Maximum` – najveća vrijednost kliznika kontrole
- `SmallChange` – korak vrijednosti za male pomake kliznika
- `LargeChange` – korak vrijednosti kad korisnik klikne unutar kontrole
- `Value` – vrijednost kontrole, predstavlja položaj kliznika
- `TickFrequency` – broj vrijednosti za koje se postavlja oznaka vrijednosti
- `TickStyle` – položaj oznaka vrijednosti

## ❑ Primjer: GUI \ Dodatak \ Trackbars

# DomainUpDown

## □ **DomainUpDown**

### ■ Svojstva:

- `Items` - kolekcija objekata s vrijednostima
- `SelectedItem` – referenca na odabrani objekt
- `Sorted` – *bool* oznaka sortiranih vrijednosti
- `Text` – sadržana vrijednost

### ■ Događaj `SelectedItemChanged`

## □ **Primjer:** **GUI \ Dodatak \ DomainUpDown**

# Događaji pri radu mišem

## ❑ Mouse Events (Delegate EventHandler, EventArgs e)

- `MouseEnter` – značka miša ulazi u područje kontrole
- `MouseHover` – značka lebdi iznad kontrole
- `MouseLeave` – značka miša izlazi iz područja kontrole

## ❑ Postupak `PointToClient(MousePosition)`

- računa koordinate značke u odnosu na kontrolu, pri čemu
- `MousePosition` - položaj značke u koordinatama zaslona

## ❑ Mouse Events (Delegate MouseEventArgs e)

- `MouseDown` – pritisnut gumb miša u području kontrole
- `MouseMove` – pomicanje miša unutar područja kontrole
- `MouseUp` – gumb miša otpušten unutar područja kontrole
- `MouseWheel` – okrenut kotačić miša
- *MouseEventArgs* razred ima svojstva
  - `Button` – pritisnuti gumb miša (left, right, middle or none)
  - `Clicks` – broj klikova The mouse pressed
  - `Delta` – broj koraka okretanja kotačića miša
  - `X, Y` – koordinate događaja, relativno u odnosu na komponentu

## ❑ Primjer: GUI \ Dodatak \ MouseEvents

# NumericUpDown

## □ NumericUpDown

### ■ Svojstva:

- `UpDownAlign` – položaj upravljačkih *Button* kontrola
- `TextAlign` – poravnanje prikazanog sadržaja
- `Minimum` – najmanja vrijednost kontrole
- `Maximum` – najveća vrijednost kontrole
- `Increment` – korak vrijednosti
- `DecimalPlaces` – broj prikazanih decimala
- `ThousandSeparator` – prikaz oznake za tisućice
- `Value` – sadržana vrijednost

### ■ Događaj `ValueChanged`

# Timer

❑ **Kontrola koja izaziva događaj u korisnički definiranim intervalima**

❑ **Svojstva**

- `Interval` – vrijeme (ms) između 2 otkucaja, vrijednost mora biti  $\geq 1$ 
  - broj sekundi proteklih između 2 otkucaja izračunava se kao  $\text{Interval}/1000$ 
    - $\text{Interval} = 100$  – mjeri desetinke,  $\text{Interval} = 10$  – mjeri stotinke
  - izraz  $1000/\text{Interval}$  = broj manjih vremenskih jedinica u sekundi
- `Enabled` – `true`:pokrenut, `false`:zaustavljen

❑ **Događaj**

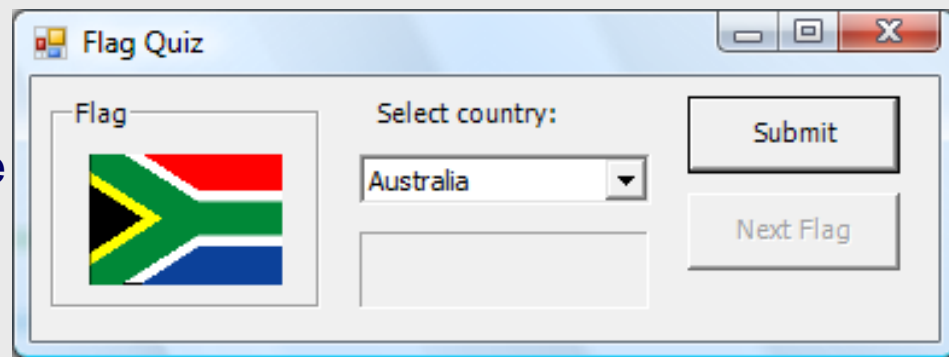
- `Tick` – otkucaj

❑ **Primjer:**  **GUI \ Dodatak \ Timer**

# Zadaci za vježbu

# Zadaci za vježbu


- ❑ **Napraviti zaslonsku masku iz vlastitog problemskog područja**
  - Ugraditi svojstva, postupke i obradu događaja, uključujući provjeru valjanosti podataka (validaciju).
  - Po potrebi napraviti vlastitu kontrolu (User control).
- ❑ **Napraviti formu za uređivanje slike**
  - Mogućnosti mijenjanja boje, svjetline, ...
  - Vrijednosti su mijenjaju uz pomoć NumericUpDown kontrole
  - Omogućiti spremanje promijenjene slike
- ❑ **Doraditi formu Partner radio gumbima za različite postavke rastezanja slike.**
- ❑ **Napraviti formu za pogađanje države čija je zastava prikazana**
  - Odabir u padajućem izborniku
  - Poruka o uspješnosti
  - Primjer  GUI \ Dodatak \ Zastave
- ❑ **Napraviti formu za unos artikla**
  - Artikl ima šifru, naziv, cijenu i sliku
  - Validirati podatke prije “spremanja”

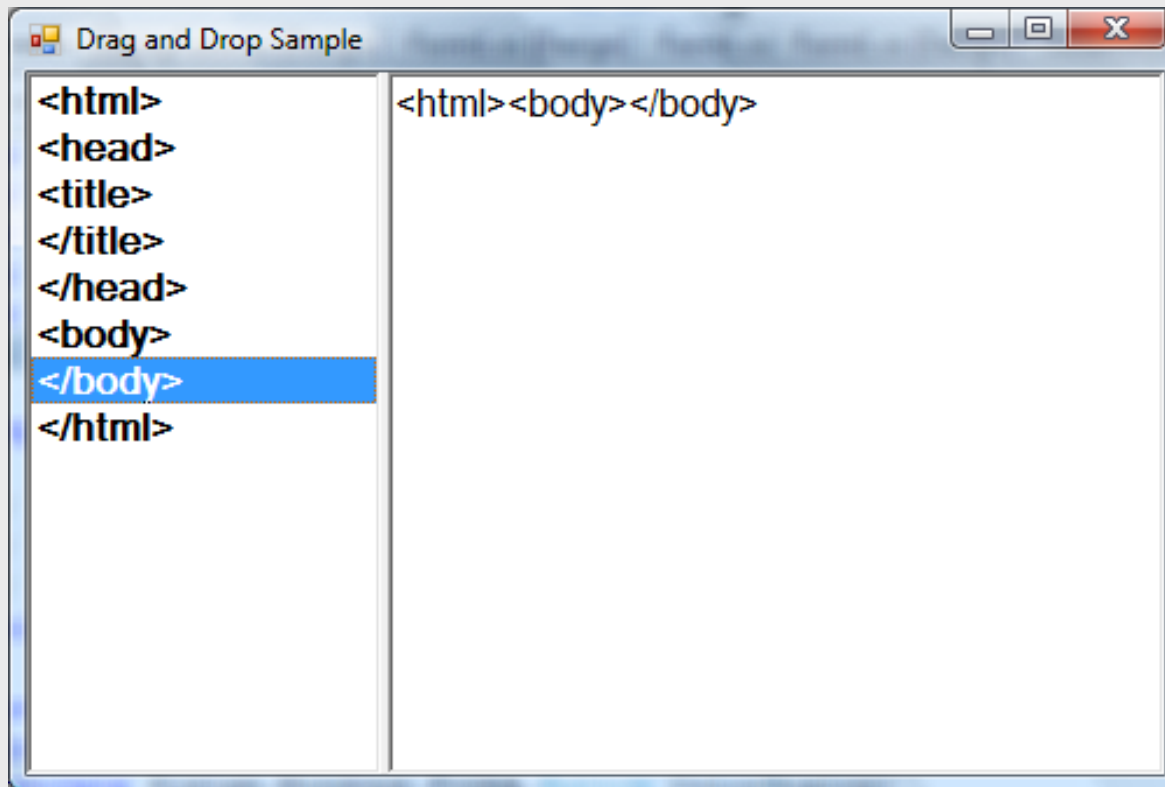




# Kako napraviti drag & drop ?

## ❑ Napraviti formu kao na slici

- S lijeve strane nalazi lista html oznaka (*tagova*)
- S desne nalazi se *TextBox* u koji će se zapisati oznake koje se mišem dovuku i ispuste (*drag&drop*) iz liste s lijeva
- Primjer:  GUI \ Dodatak \ Html



# Automatizirano povlačenje i ispuštanje (1)

- ❑ Automatizirano povlačenje i ispuštanje (Drag&Drop)
- ❑ Podrazumijeva se postojanje dvije kontrole
  - izvor (drop source)
  - cilj (drop target)
- ❑ Cilj mora imati svojstvo
  - `AllowDrop` – postavljeno na `true`
- ❑ Postupak kojim izvor inicira Drag&Drop operaciju
  - `DoDragDrop(object data, DragDropEffects allowedEffects );`
  - pri tome najavljuje dozvoljenu posljedicu

```
enum DragDropEffects {  
    Copy, // Take a copy of the data  
    Move, // Take ownership of the data  
    Link, // Link to the data  
    Scroll, // Scrolling is happening in the target  
    All, // All of the above  
    None, // Reject the data  
}
```

# Automatizirano povlačenje i ispuštanje (2)

## ❑ Događaji (object sender, DragEventArgs e)

- DragEnter - miš ulazi u područje cilja, koji dojavljuje da li prihvaća podatke
- DragOver - prolazak miša preko cilja
- DragLeave - miš napušta područje cilja
- DragDrop - podaci ispušteni na cilj

## ❑ Članovi razreda DragEventArgs

- AllowedEffect – učinci koje dozvoljava izvor
- Data – podaci koji se prenose
  - GetDataPresent – postupak kojim se provjerava da li podaci odgovaraju željenom tipu
  - GetData – postupak uzimanja podataka u željenom tipu
- Effect – postavljanje ili uzimanje učinka na cilju
- KeyState – oznaka da je pritisnuta neka od upravljačkih tipki
- X, Y – screen koordinate značke

# Primjer višenitnog programa

---

# Višenitnost

## ❑ Nit ili dretva (eng. *thread*)

- Operacijski sustav razdvaja izvođenje aplikacija u procese
- Jedan proces može imati više niti izvođenja
- Niti su osnovne jedinice u kojima OS raspodjeljuje procesorsko vrijeme
- Ime niti dolazi od 'thread of execution'.

## ❑ Višenitnost (*multithreading*)

- Više niti može se izvoditi unutar jednog procesa
- Višenitni program simulira istovremeno izvođenje različitih niti (pseudo-paralelizam).

## ❑ Primjer:

- poslužitelj (web, baze podataka), koji "istovremeno" obrađuje zahtjeve klijenata

## ❑ Prednost

- Bolja iskoristivost procesorskog vremena, naročito kod programa sa sporim periferijama

## ❑ Nedostatak

- Složenije programiranje, problem sinkronizacije niti

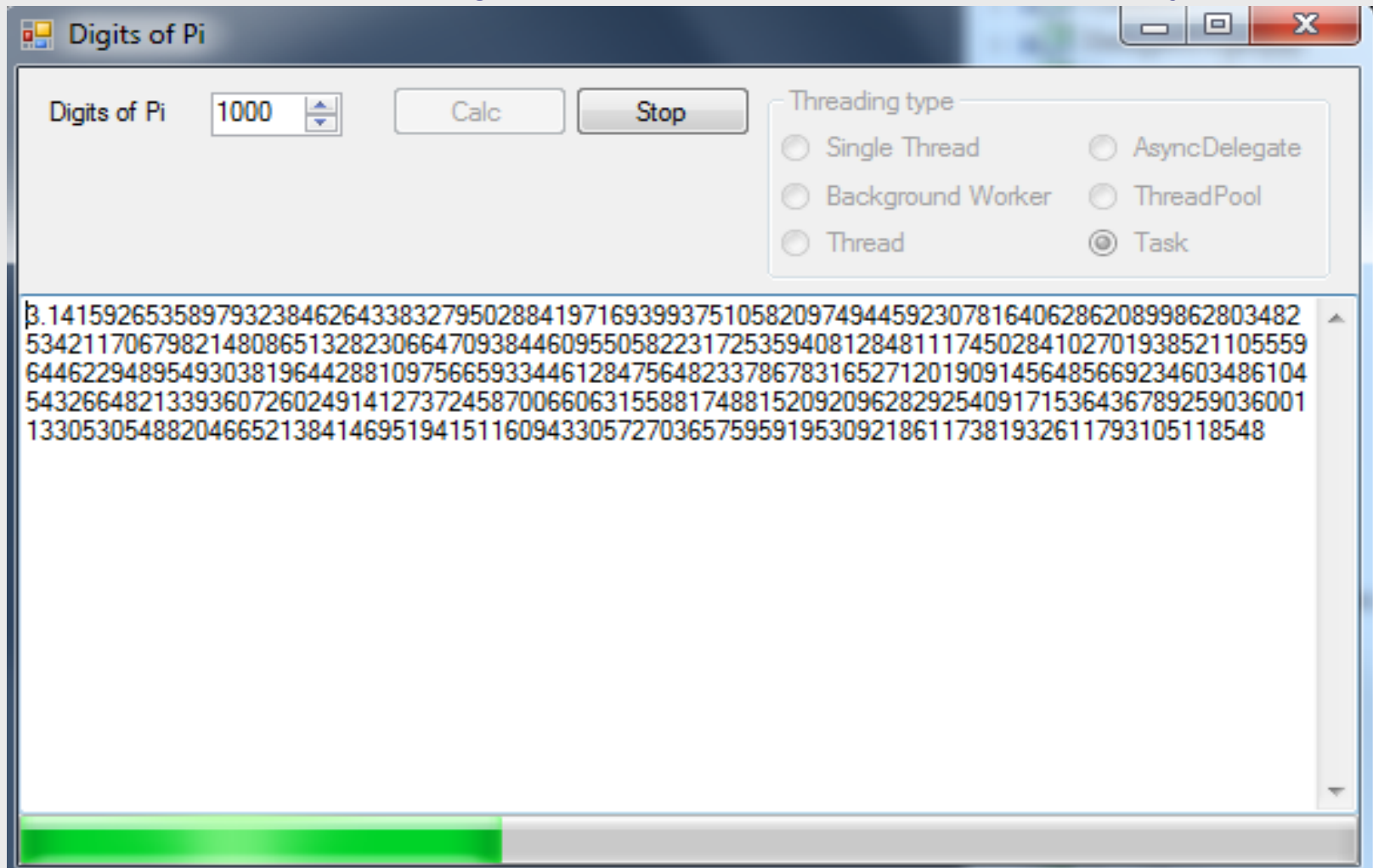
# Višenitnost i paralelno programiranje u C#-u

- ❑ **Asinkrono pozivanje postupaka**
  - Svaki 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: korištenje asinkronih verzija postupaka (ako takve postoje za određenu namjenu) + `async` i `await`**

# Primjer višenitnog programa (1)

## ❑ Primjer GUI \ Dodatak \ RacunanjePI\_ViseNacina

- problem istovremenog prikaza rezultata i statusa za veći broj znamenki



# Primjer višenitnog programa (2)

## ❑ Primjer GUI \ Dodatak \ RacunanjePi\_ViseNacina \ SimplePiCalculator.cs

- Postupak koji treba obaviti u pozadini

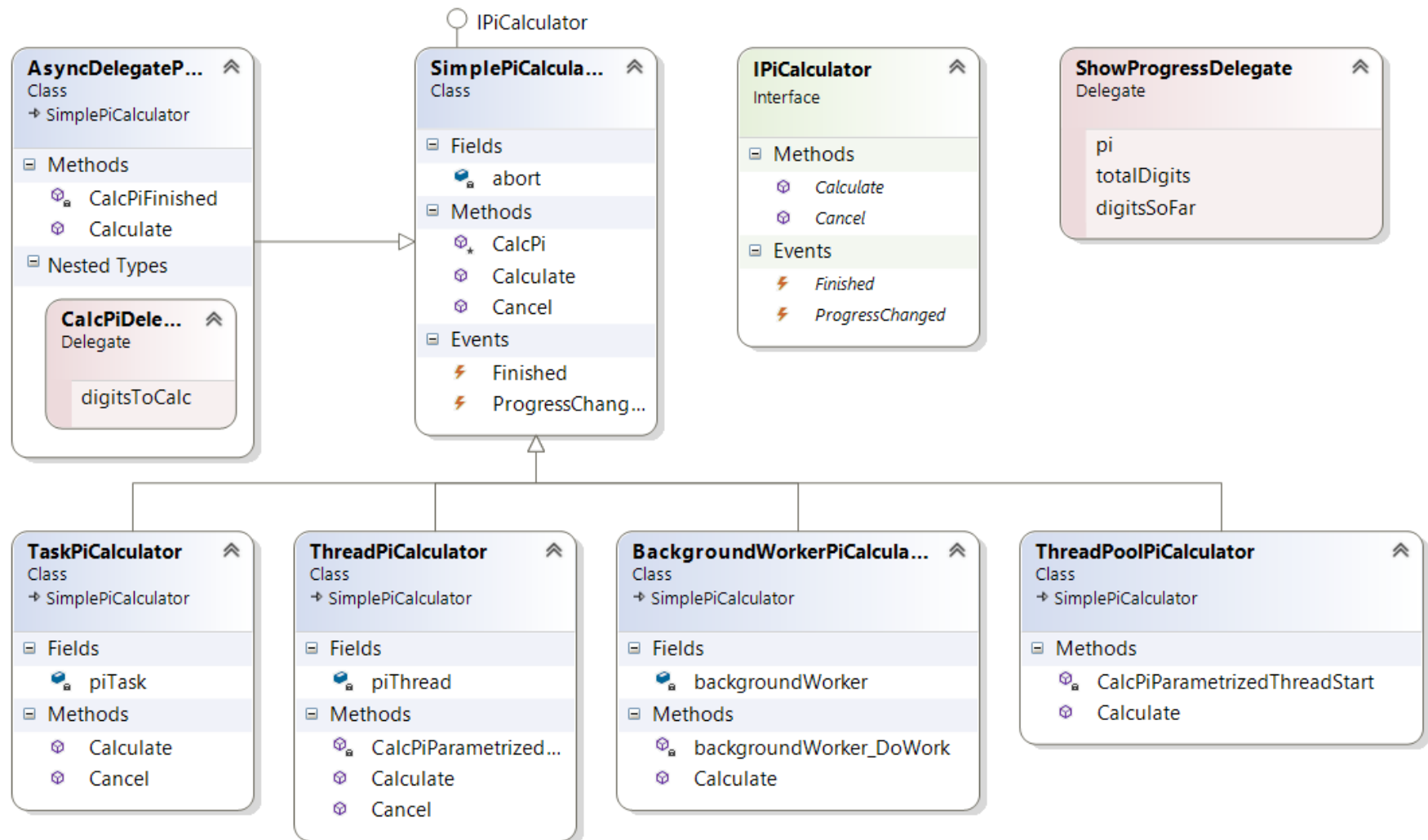
```
void CalcPi(int digits) {  
    ...  
    for (int i = 0; i < digits; i += 9) {  
        //računanje sljedećih 9 decimala  
        ...  
        //prikaz trenutnog stanja podizanjem vlastitog događaja  
        ProgressChanged(pi.ToString(), digits, i + digitCount); ...  
        //provjera da li je dan zahtjev za prekid postupka  
        ...  
    }  
    ...  
    //podizanje vlastitog događaja za kraj računanja  
    Finished(this.GetType().Name);  
}
```

- Za provjeru prekida postupka primjer koristi zastavicu (boolean varijabla)
- Od verzije .NET 4 umjesto zastavice koristi se struktura CancellationToken



# Primjer višenitnog programa (3)

- Različiti načini pozivanja konkretnog postupka *CalcPi*
- Sve varijante implementiraju sučelje *IPiCalculator*
- Forma pretplaćena na događaje iz *IPiCalculatora*



# Asinkroni poziv postupka

## ❑ Primjer GUI\Dodatak\RacunanjePi\_ViseNacina\AsyncDelegatePiCalculator.cs

- Definira se novi tip delegata prema obliku postojećeg postupka

```
private delegate void CalcPiDelegate(int digitsToCalc);
```

- Inicijalizira se delegat te se na njemu poziva postupak *BeginInvoke* s parametrima
  - parametri postupka koji se poziva
  - postupak koji se želi pozvati po završetku izvođenja
  - objekt koji se želi prenijeti postupku koji će se izvršiti po završetku
- Postupak se odvija u posebnoj, pozadinskoj niti
- Po okončanju postupka dohvat (eventualnih) vrijednosti vrši se s *EndInvoke*

```
CalcPiDelegate calcPiMethod = CalcPi;  
IAsyncResult result = calcPiMethod.BeginInvoke(digitsToCalc,  
                                                CalcPiFinished, null);  
  
...  
private void CalcPiFinished(IAsyncResult asyncResult) {  
    CalcPiDelegate calcPiMethod = (CalcPiDelegate) ((AsyncResult)  
                                                asyncResult).AsyncDelegate;  
    calcPiMethod.EndInvoke(asyncResult);  
}
```

# Kontrola *BackgroundWorker*

## ❑ Primjer GUI\Dodatak\RacunanjePi\_ViseNacina\BackgroundWorkerPiCalculator.cs

- Postupak *RunWorkerAsync* uzrokuje izvođenje koda pridruženog događaju *DoWork* u pozadinskoj niti
- Niti se može poslati samo objekt - dohvat se vrši preko svojstva *Argument*

```
backgroundWorker = new BackgroundWorker();  
backgroundWorker.DoWork += backgroundWorker_DoWork;  
backgroundWorker.RunWorkerAsync(digitsToCalc);  
  
private void backgroundWorker_DoWork(object sender,  
                                     DoWorkEventArgs e) {  
    CalcPi((int)e.Argument);  
}
```

- Ako je potrebno rezultat se može pospremiti u *e.Result*

## ❑ **BackgroundWorker** još definira događaje:

- za kraj: *RunWorkerCompleted* s parametrom *Result* u koji se može pospremiti rezultat
- za obradu događaja *ProgressChanged* - inicira se pozivom postupka *ReportProgress*
- *Napomena: navedeni događaji nisu korišteni u ovom primjeru*

# Stavljanje postupka u red niti( GUI\RacunanjePi)

## ❑ Primjer GUI\Dodatak\RacunanjePi\ThreadPoolPiCalculator.cs

- Statički postupak *QueueUserWorkItem* u razredu *ThreadPool*
- Potreban delegat tipa *WaitCallback*
  - *WaitCallback* je tip delegata kojem odgovara postupak koji prima jedan argument tipa *object* i ne vraća ništa.
  - Prilikom stavljanja postupka u red niti može se proslijediti samo jedan objekt

```
public class ThreadPoolPiCalculator : SimplePiCalculator {  
    public override void Calculate(int digitsToCalc) {  
        ThreadPool.QueueUserWorkItem(  
            CalcPiParametrizedThreadStart, digitsToCalc);  
    }  
    private void CalcPiParametrizedThreadStart(object o) {  
        CalcPi((int)o);  
    }  
}
```

- Postupak započinje u trenutku kad postoji slobodna pozadinska nit
- Broj niti po procesu može se saznati s *ThreadPool.GetMaxThreads* i promijeniti s *ThreadPool.SetMaxThreads*

# Stvaranje nove niti (GUI\RacunanjePi)

- ❑ **Primjer** GUI\Dodatak\RacunanjePi\_ViseNacina\ThreadPiCalculator.cs
- ❑ **Da bi se stvorila nova nit, treba instancirati objekt tipa *Thread*.**
  - Konstruktor razreda *Thread* prima delegat tipa *ThreadStart* (*void* postupak bez argumenata) ili tipa *ParametrizedThreadStart* (*void* postupak s jednim argumentom tipa *object*)
- ❑ **Stvorena nit pokreće se pozivom postupka *Start***
  - Pri pozivu se može poslati jedan objekt

```
private void CalcPiParametrizedThreadStart(object o) {  
    CalcPi((int)o);  
}  
...  
piThread = new Thread(CalcPiParametrizedThreadStart);  
piThread.IsBackground = true;  
piThread.Start(digitsToCalc);
```

- Ako svojstvo *IsBackground* nije postavljano na *true* novostvorena nit je glavna nit (eng. foreground thread)
- ❑ **Program se izvršava sve dok postoji barem jedna glavna nit**
- ❑ **Pozadinske niti automatski završavaju završetkom programa**

# Stvaranje novog zadatka (📁 GUI\RacunanjePi)

❑ Primjer 📁 GUI\Dodatak\RacunanjePi\_ViseNacina\TaskPiCalculator.cs

❑ Koriste se razredi *Task* i/ili *Task<TResult>*

- Predstavljaju postupke koji se odvijaju u pozadini a ne vraćaju vrijednosti odnosno vraćaju vrijednost 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 jednim argumentom tipa *object* koji vraća objekt tipa *TResult*
- Ostali (opcionalni) parametri konstruktora uključuju argument koji se proslijeđuje postupku te razred za signalizaciju otkazivanja zadatka

❑ Napomena:

- Umjesto eksplicitno definiranih delegata uobičajeno je koristiti lamda izraze

```
piTask = new Task(() => CalcPi(digitsToCalc));  
piTask.Start();
```

# Pristup kontrolama korisničkog sučelja iz drugih niti

## ❑ Primjer GUI\Dodatak\RacunanjePi\_ViseNacina

- 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){
    // da li je poziv došao izvan primarne niti
    if (this.InvokeRequired){
        ShowProgressDelegate showProgress = ShowProgress;
        this.BeginInvoke(showProgress, pi, totalDigits,
                        digitsSoFar);
    }
    else{
        piTextBox.Text = pi;
        piProgressBar.Maximum = totalDigits;
        piProgressBar.Value = digitsSoFar;
    }
}
```

# Još malo o nitima ...

## ❑ Postupci

- Statički postupak `Sleep` - uzrokuje privremeno zaustavljanje izvršavanja niti iz koje je pozvana za navedeni broj milisekundi.
- `Thread.Sleep(n)`
- `Thread.Suspend()` – privremeno zaustavlja izvršavanje niti
- `Thread.Resume()` – ponovno pokreće privremeno zaustavljenu nit
- `Thread.Abort()` – uzrokuje `ThreadAbortException` na niti na kojoj se izvrši, što uzrokuje završetak niti
  - Ne preporuča se koristiti `Abort`, već “obavijestiti” nit da je vrijeme za završetak
- `Join()` – postupak kojim se u trenutnoj niti čeka na završetak niti na kojoj je pozvan postupak

## ❑ Svojstva

- svojstvo `Priority` - Postavljanje prioriteta pojedine niti
  - `public ThreadPriority Priority{ get; set; }`
- svojstvo `ThreadState` - stanje niti
- `Thread.CurrentThread` - referenca na trenutnu nit



# Prekidanje pozadinskih niti, provjera stanja i dohvat vrijednosti (1)

- ❑ Čak i ako postoje ugrađeni mehanizmi za otkazivanje pozadinskih postupaka, preporuka je koristiti **CancellationToken** i **CancellationTokenSource**

- [http://msdn.microsoft.com/en-us/library/vstudio/dd997364\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/dd997364(v=vs.110).aspx)

- ❑ **Asinkrono pozivanje postupaka**

- Sučelje *IAsyncResult* kao rezultat poziva *BeginInvoke*
  - Svojstvo *IsCompleted* za provjeru stanja
  - Svojstvo *AsyncWaitHandle* za objekt tipa *WaitHandle* (mehanizam za čekanje na dijeljeni resurs)
- Dohvat povratne vrijednosti pozivom postupka *EndInvoke*
- Nema direktnog mehanizma za otkazivanje asinkronog poziva

- ❑ **Kontrola *BackgroundWorker***

- Svojstvo *IsBusy* za provjeru izvodi li se postupak
- Dohvat povratne vrijednosti preko argumenta u događaju *RunWorkerCompleted*
- Namjera prekidanja aktivnog postupka postupkom *CancelAsync*
  - Postupak može provjeravati vrijednost *CancellationPending* i sam dovršiti

izvođenje

# Prekidanje pozadinskih niti, provjera stanja i dohvat vrijednosti (2)

## ❑ Niti u redu niti (razred *ThreadPool*)

- Nema direktnih mehanizama provjere stanja, otkazivanja i dohvata vrijednosti

## ❑ Niti stvorene razredom *Thread*

- Provjera stanja pomoću svojstva *ThreadState*
- Čekanje na dovršetak postupkom *Join*
- Nema direktnog mehanizma za dohvat povratne vrijednosti
- Prekidanje niti može se izvesti postupkom *Abort*, ali se ne preporuča

## ❑ Niti stvorene pomoću zadatka (razred *Task*)

- Provjera stanja pomoću svojstva *Status*
- Čekanje na dovršetak postupkom *Wait*
- Povratna vrijednost u svojstvu *Result*
- Namjera prekidanja postupka se vrši pomoću objekta tipa *CancellationToken* predanog u konstruktoru
  - U kodu koji se izvršava za neki zadatak potrebno provjeriti da li je dan nalog za otkazivanje zadatka