

Korisničko sučelje i dijalozi. Datoteke i serijalizacija

2014/15.07

Korisničko sučelje i dijalozi

Osnovni pojmovi

❑ Sučelja aplikacije

- korisničko sučelje (user interface) – definira interakciju s krajnjim korisnikom
- sučelje sustava, sistemsko sučelje (system interface) – određuje način razmjene informacija s drugim sustavima

❑ Osnovni mehanizmi korisničkih sučelja

- navigacijski mehanizam (navigation mechanism) – osigurava način na koji korisnici određuju što žele napraviti
- ulazni mehanizam (input mechanism) – određuje način prihvata informacija
- izlazni mehanizam (output mechanism) – određuje način pružanja informacija korisnicima ili drugim sustavima

❑ Načela oblikovanja korisničkog sučelja

- Raspored (Layout)
- Uvažavanje sadržaja (Content awareness)
- Estetika (Aesthetics)
- Iskustvo korisnika (User experience)
- Dosljednost, konzistentnost (Consistency)
- Lakoća korištenja, minimalni napor korisnika (Minimal user effort)

Raspored, organizacija sučelja

☐ Programska oprema mora imati standardan izgled zaslona.

- područje za navigaciju - izbornici (najčešće na vrhu)
- područje za prikaz statusa obrade i podataka (najčešće na dnu)
- radno područje - rad s podacima (u sredini)

☐ Polja zaslonske maske moraju biti logički grupirana.

- npr. osobni podaci, podaci o adresama i kontaktima, podaci o članovima obitelji, podaci o školovanju i napredovanju
- raspoređivanje na različite zaslonske maske (forme, prozore)
- grupiranje spremnicima (container) - panelima, tabulatorima
- učajurivanje u vlastite korisničke kontrole

☐ Poravnanje

- vertikalno i horizontalno poravnanje kontrola
- lijevo poravnanje polja za unos i (desno) poravnanje njihovih labela
- lijevo poravnanje nizova znakova, desno poravnanje brojeva

Uvažavanje sadržaja

- ☐ **U svakom trenutku mora biti vidljiva informacija o**
 - dijelu obrade (unos, izmjena ...),
 - vrsti prikazanih podataka (osoba, račun, ...),
 - količini podataka (zapis m od n),
 - mogućim akcijama (aktivne kontrole).

- ☐ **Elementi sučelja trebaju se odnositi na poslovne objekte, a ne na tehničke aspekte aplikacije.**

- ☐ **Objekti sučelja trebaju oponašati izgled i ponašanje stvarnih objekata koje prikazuju (npr. dokumente)**

Estetika

☐ Preglednost podataka

- Izbjegavati sažimanje kontrola na uskom prostoru
- Minimizirati neiskorišten prostor
- Minimizirati prekrivanje (npr. u slučaju više prikazanih "prozora").

☐ Korišćenje pisma

- Koristiti najviše 4 različite veličine slova na ekranu (web šareniji)
- Koristiti najviše 3 vrste pisma (font) na jednom ekranu
- Koristiti kombinaciju velikih i malih slova (izbjegavati kapitalizaciju)

☐ Upotreba boja

- Koristiti najviše 4 različite boje na ekranu
- Paziti na kontrast – tamni tekst na svijetloj podlozi i obrnuto
- Bojanje neaktivnih kontrola npr. u sivo, bolje je nego sakrivanje kontrola
- Ne koristiti plavu boju za tekst (teško se čita i uobičajena je za poveznice)

Prilagođenost korisniku

☐ Prilagođenost iskustvu korisnika

- lakoća učenja korištenja od strane početnika
- lakoća korištenja od strane naprednog korisnika
- podrška naprednim korisnicima, npr. ubrzavajućim tipkama (accelerator key)

☐ Svakako ugraditi interaktivnu pomoć ovisnu o kontekstu.

☐ Korisnike treba navoditi na služenje programom

- najčešće interaktivnom pomoći i uputama (uz dosljednost)
- manja potreba za uputama → intuitivnost → kvalitetnije sučelje

☐ Sučelje mora imati ujednačene standardne poruke

- Poruke moraju biti jednostavne, precizne te ovisne o kontekstu.
- Izbjegavati računalski žargon ikratice.
- Prevesti poruke na stranom jeziku (npr., iznimke OS i SUBP)

Dosljednost korisničkog sučelja

☐ Dosljednost (konzistentnost) korisničkog sučelja

- kontrole istog tipa svuda jednakog izgleda i ponašanja
- standardne kontrole uvijek na uobičajenom/očekivanom mjestu (gumbi, status, ...)

☐ Najvažniji faktor za pojednostavnjivanje načina korištenja sustava

- dosljednost sučelja omogućuje korisnicima predvidjeti što će se dogoditi
- nakon što svladaju interakciju s jednim dijelom sustava, znat će i s drugim dijelovima.
- dosljednost skraćuje krivulju učenja

☐ Postavljanje i pridržavanje standarda (slično standardu kodiranja)

- izgled (veličina – omjer, oblik, boja),
- naslovi kontrola
- ponašanje kontrola
- standardizacija tipki

Minimizacija napora

☐ **Pravilo tri klika (three clicks rule)**

- Treba omogućiti pristup od izbornika do podatka u najviše tri koraka (klika mišem ili tipkovničkih kombinacija)

☐ **Unos se mora obavljati u slijedu kojim su polja fizički poredana.**

- slijeva nadesno i
- postavljanjem *TabIndex* svojstva kontrole

☐ **Korištenje predviđenih (default) vrijednosti**

☐ **Predviđene (default) kontrole**

- Nedestruktivne - npr. "Ne" za brisanje


☐ **Ograničavanje selekcije listama**

- preddefinirani skupovi podataka
- vezani podaci (strani ključevi)


☐ **Ulančavanje procedura (primjer u nastavku)**

Vrste korisničkog sučelja

❑ Sučelje s pojedinačnim dokumentima - Single document interface (SDI)

- sučelje sastavljeno od nezavisno otvorenih i odvojenih formi, od kojih svaka sadrži po jedan "dokument"
- primjer:  ADO\EF_Firma\SDIForm

❑ Sučelje s više dokumenata - Multiple document interface (MDI)

- aplikacija se sastoji od jedne ili više formi sadržanih unutar zajedničkog glavnog prozora
- primjer:  ADO\EF_Firma>MainForm

❑ Sučelje s tabuliranim dokumentima - Tabbed document interface (TDI)

- različiti dokumenti sadržani unutar jednog prozora, na panelima Tab kontrole
- primjer: Visual Studio razvojna okolina

Izbornici i dijalozi

❑ Sučeljem moraju biti podržane različite vrste izbornika

- horizontalni izbornik (menu bar) - uvijek vidljiv, lako dohvatljiv
- padajući (pull-down) i kaskadni - "nevidljivi", ali se opcije daju grupirati
- skočni (pop-up, brzi) - nije očigledno da postoje, skaču na različitim mjestima
- trake s ikonama (toolbar, toolbox, ribbon) - vidljivi, pamtljivi, uz mogući problem dinamičkog osposobljavanja i prikaza/skrivanja ikona

❑ Omogućiti brzi odabir

- ubrzavajućom/funkcijskom tipkom ili slovom opcije izbornika
 - primjer: Alt-Slovo kombinacija pisanjem "&" ispred željenog slova
- Tipke za obavljanje standardnih funkcija moraju biti definirane pažljivo i jednoznačno
 - primjer: F1-Pomoć, F2-Unos, F3-Izmjena, ESC-Opoziv)
 - Unaprijed treba predvidjeti i one za aktiviranje dodatnih funkcija.

❑ Ne pretjerivati sa širinom i dubinom izbornika!

❑ Konzistentan, pozitivan tekst poruka, punim rečenicama !

Kriterij za odabir kontrola grafičkog sučelja

☐ Text Box i varijante (npr. RichTextBox)

- unos slobodnog teksta
- tekst može biti maskiran (npr. telefonski broj, registarska oznaka)
- omogućiti unos više redaka i kliznike samo za stvarno višeretkovni tekst (npr. opis, komentar)
- pripaziti na duljinu teksta i tip podatka

☐ Check Box

- binarne vrijednosti, opcionalno "nepoznato"
- mali broj (do 3) unaprijed poznatih i vremenski nepromjenjivih oznaka
- za veći broj oznaka, koristiti listu (List Box) sa selekcijom više stavki

☐ Radio Button (ili Option Button)

- mali broj (do 3) unaprijed poznatih i vremenski nepromjenjivih oznaka
- vrijednosti se međusobno isključuju
- za veći broj oznaka, koristiti listu za odabir jedne vrijednosti (Combo Box)

Kriterij za odabir kontrola grafičkog sučelja

❑ Drop-Down Box (ili Combo Box)

- umjereno velik broj (do nekoliko stotina), međusobno isključivih vrijednosti
- do 20 istovremeno vidljivih vrijednosti pri odabiru
- u slučaju većeg broja zapisa koristiti editabilni *combo* za unos filtra ili kontrolu nadomjestiti posebnom formom za filtriranje i odabir (tzv. lookup)

❑ List Box

- umjereno velik broj (nekoliko desetaka), ne nužno isključivih vrijednosti
- istovremeni prikaz 8-10 vrijednosti
- u slučaju većeg broja zapisa ili zapisa složenije strukture nadomjestiti rešetkom (Grid)

❑ NumericUpDown

- nevelik slijed (nekoliko desetaka) diskretnih vrijednosti
- u slučaju većeg broja vrijednosti, omogućiti izravan upis ili nadomjestiti listom

❑ Grid – rešetka, mreža, matrica

- kombinacije osnovnih elemenata
- koristiti za prikaz zapisa s malim brojem atributa da se vidi cijeli zapis odjednom

Primjer sučelja poslužiteljske aplikacije

kfertalj (ansi) | PIS - PERSONALNI INFORMACIJSKI SUSTAV | (Sri) 04.12.96

OSOBA: Dohv Sljed Preth Unos Izmj Ostalo Lista Rasp Zap ...

Postavljanje uvjeta za dohvatzapisa

===== (LALIĆ MARIJAN) ===== [(2/3)] ===== 120/261 =====

Sprema (71) (Visoka VII/1)

Br.svjed.(kfertalj (ansi) | PIS - PERSONALNI INFORMACIJSKI SUSTAV | (Čet) 05.12.96

Zanimanje (5

Stamb.stanje (3 ESC-Prihvat
CtrC-Prekid F10-Pomoć

Krvna grupa (= unos uvjeta ===== OSOBA ===== [(1/3)] =====

Zdrav.broj (ID () Prezime (*IĆ Roditelj ()

Spol () Ime (MARI* (čiji/čija) ()

Izlazak () JMBG () Br.osobnika ()

Ured za obranu

Osobni VPD -----
Početak voj.roka Kategorija osobe () ()

Vojska voj.roka Bračno stanje () () Djece ()

Razl.prest.voj.roka Mjesto rođenja () () Dat.rođenja ()

Sudjel. u dom.ratu Općina () ()

Država () ()

Narodnost () ()

Vjera () ()

Državljanstvo 1 ()

Državljanstvo 2 ()

Prebivalište () (ZAGREB) ()

Mjesto stanovanja () ()

Tel. ()

Naziv mjesta

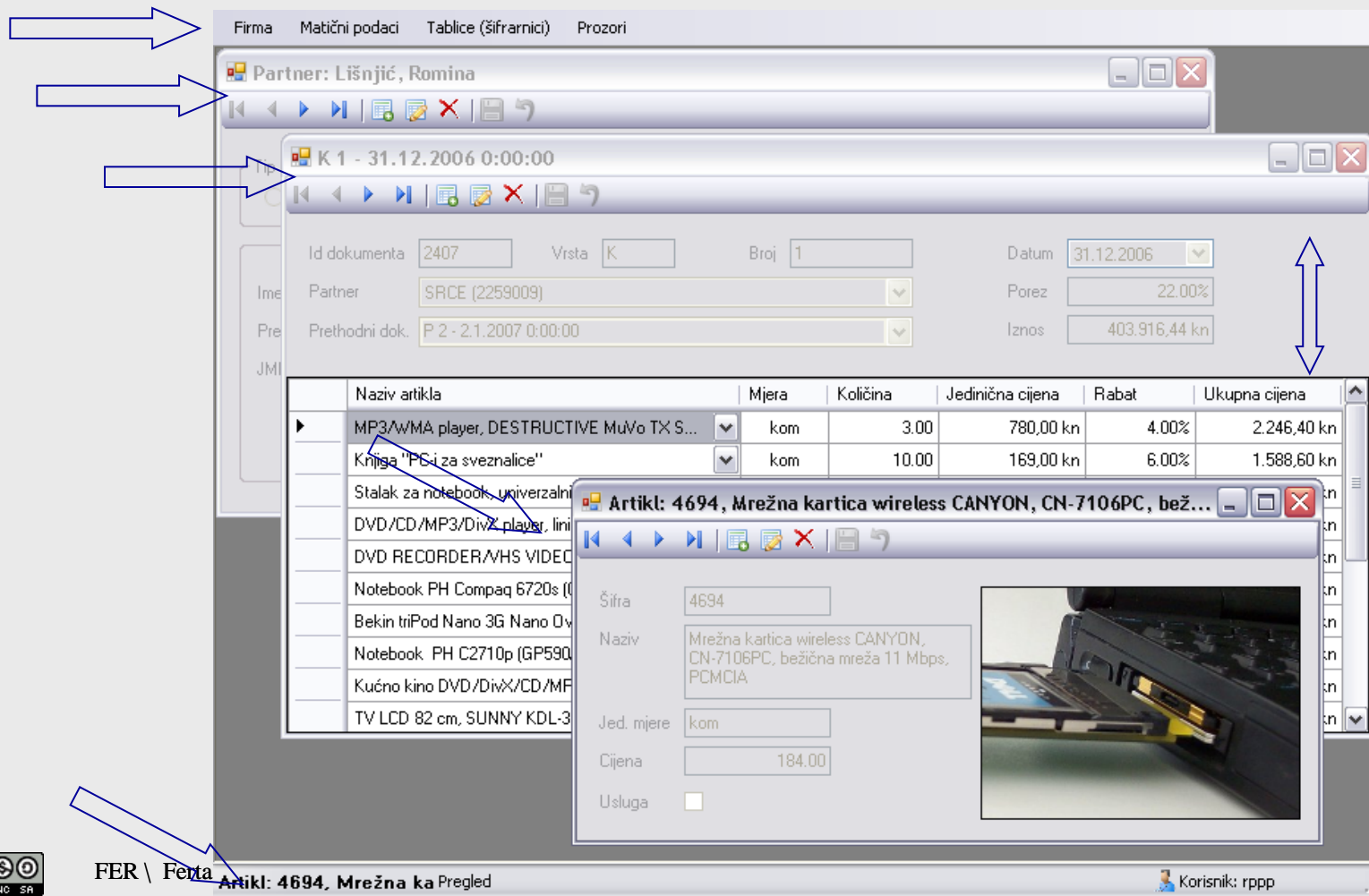
[illegible]

Primjer sučelja aplikacije

MDI korisničko sučelje

❑ MDI (Multiple Document Interface) aplikacija

- aplikacija se sastoji od formi sadržanih unutar zajedničkog glavnog prozora
- izuzetak čine modalne forme koje mogu biti otvorene povrh glavne forme



Forma roditelj i forme djeca

- ❑ **Roditelj (*MdiParent*)** – glavna forma koja sadrži druge forme
 - Svojstvo `IsMdiContainer = true` (u dizajnu)
 - `MdiChildren` – kolekcija formi djece (u pogonu)
 - `LayoutMdi` – raspored djece unutar roditeljske forme (u pogonu)

- ❑ **Dijete (*MdiChild*)** – forma sadržana u formi roditelj
 - svojstvo `MdiParent` referencira roditeljsku formu prije prikazivanja djeteta

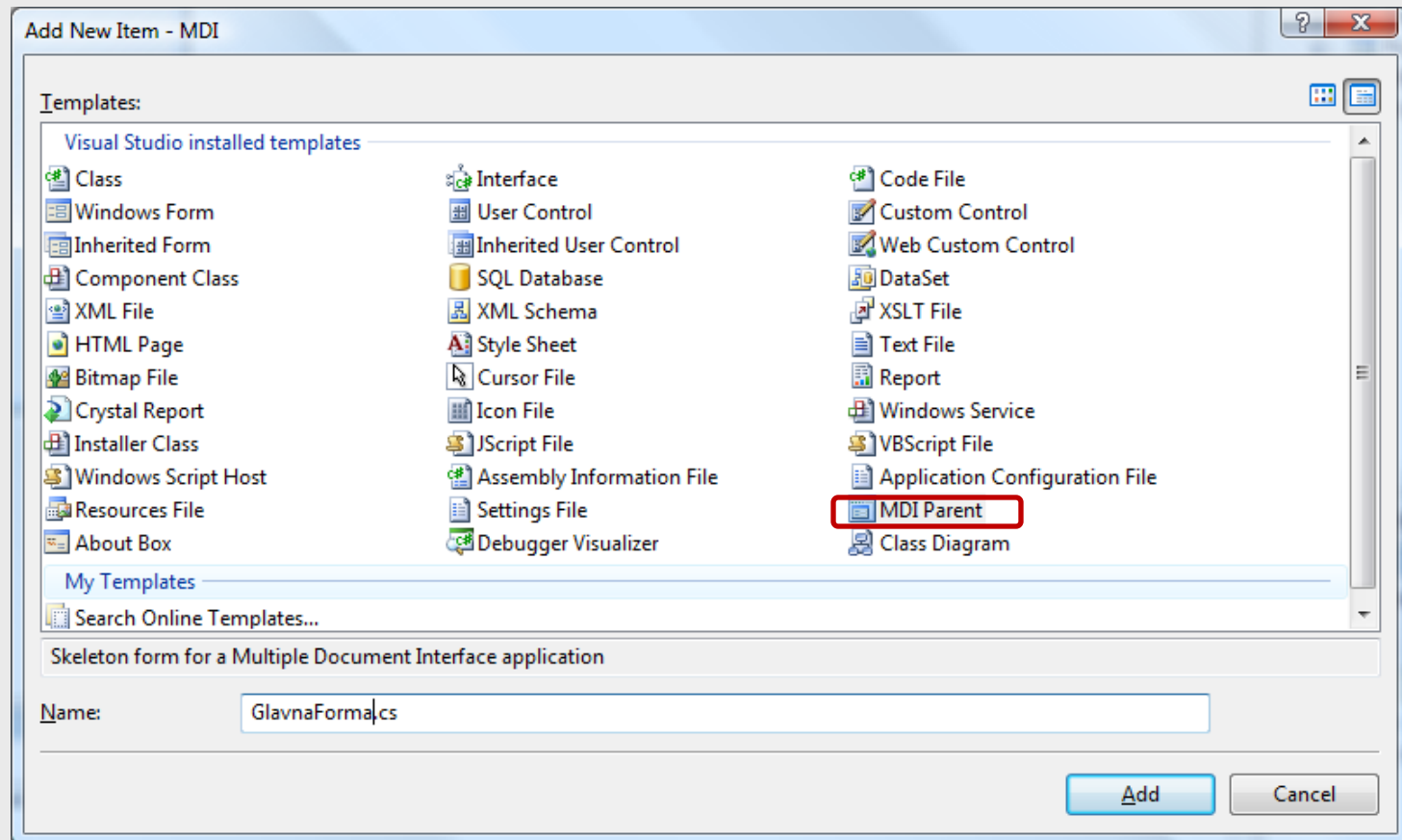
- ❑ **Primjer:**  **ADO\EF_Firma - MainForm**

```
private void artiklToolStripMenuItem_Click(...)  
{  
    ArtiklForm f = new ArtiklForm();  
    f.MdiParent = this;  
    f.Show();  
}
```

Kreiranje MDI roditelja

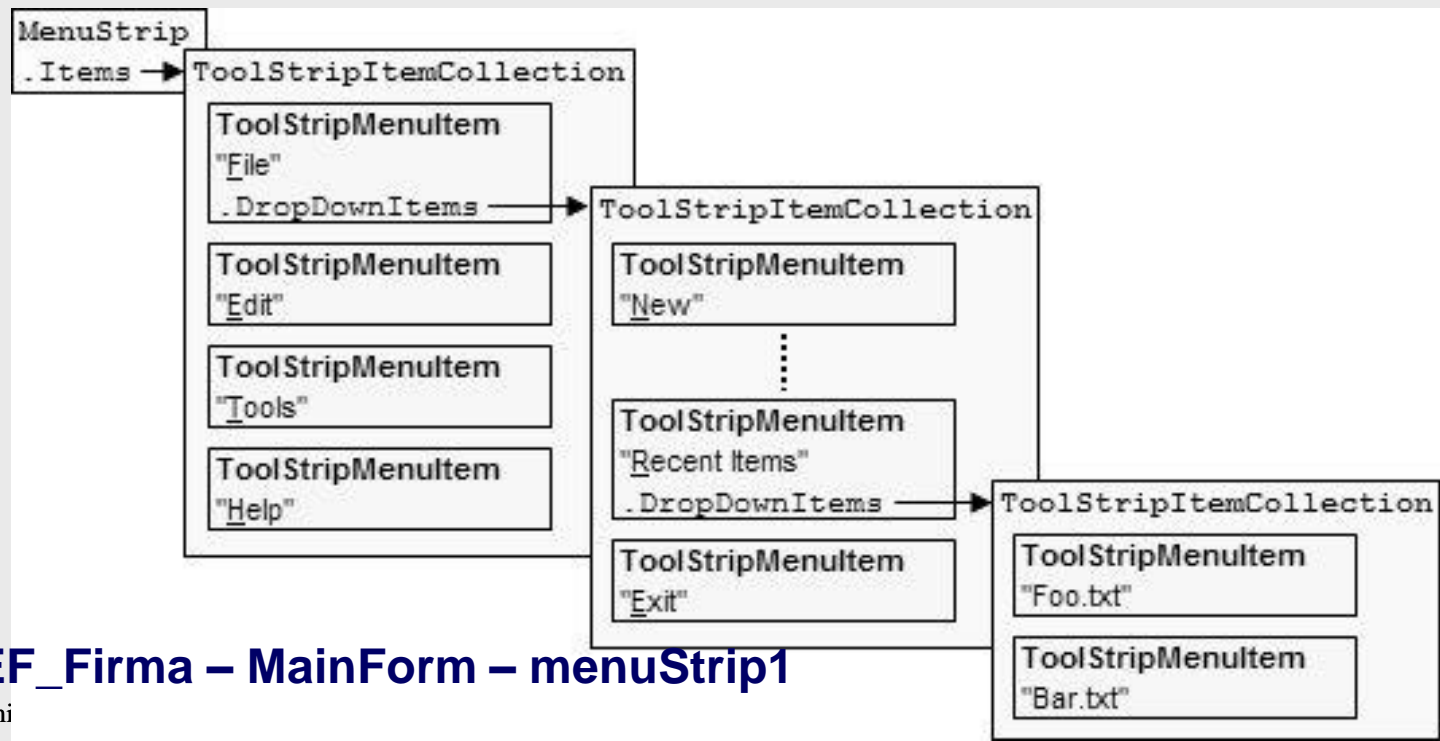
❑ Formu roditelja je moguće automatski generirati

- Dodavanjem nove datoteke *MDI parent* u projekt
- Stvaraju se izbornik i alatna traka sa standardnim mogućnostima (*open, save, copy, cut, exit, ...*)



Izbornik

- ❑ **MenuStrip** - kontrola koja predstavlja korijen sustava izbornika
 - Sadrži kolekciju *Items* tipa *ToolStripItemCollection*
- ❑ **Element izbornika može biti tipa (desni klik – Convert to)**
 - *ToolStripMenuItem* – standardni element izbornika
 - može sadržavati podelemente (svojstvo *DropDownItems*)
 - *ToolStripComboBox* – padajuća lista
 - *ToolStripTextBox* – tekst



❑ **Primjer:** ADO\EF_Firma – MainForm – menuStrip1

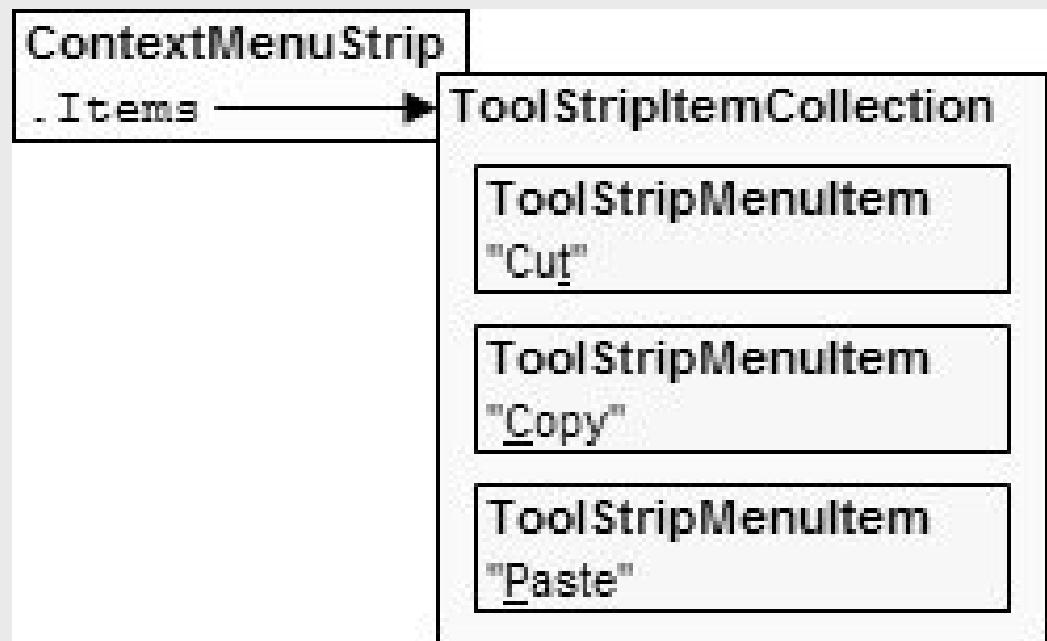
Izbornik zavisan o kontekstu

❑ **ContextMenuStrip**

- izbornik zavisan o kontekstu (brzi izbornik, skočni izbornik, izbornik prečice)
- sadrži kolekciju *Items* tipa *ToolStripItemCollection*
- može imati hijerarhiju opcija ali nema horizontalnu početnu komponentu

❑ **Forma/kontrola može imati najviše jedan izbornik konteksta**

- *ContextMenuStrip* svojstvo forme ili kontrole



Raspored djece unutar roditeljske forme

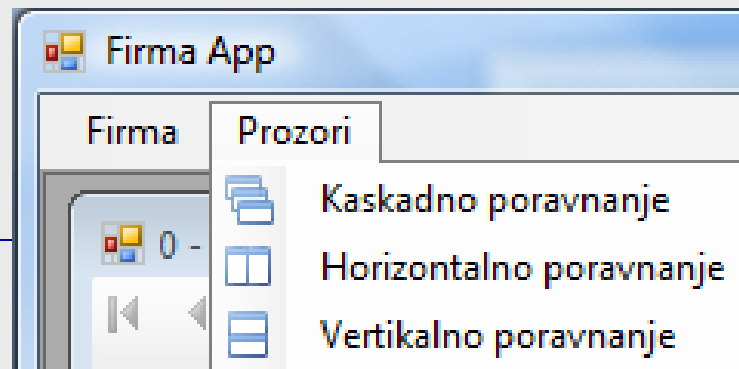
❑ Postupak `LayoutMdi` na roditeljskoj formi, s argumentom

■ enumerator `MdiLayout`

- `Cascade` – kaskadni raspored unutar roditeljske forme
- `TileVertical` – okomito poravnanje unutar roditeljske forme
- `TileHorizontal` – vodoravno poravnanje unutar roditeljske forme
- `ArrangeIcons` – poravnanje ikona formi (minimiziranih formi)

❑ Primjer: `ADO\EF_Firma - MainForm`

```
private void horizontalnoPoravnanjeToolStripMenuItem_Click(  
    object sender, EventArgs e)  
{  
    LayoutMdi (MdiLayout.TileHorizontal) ;  
}  
...
```



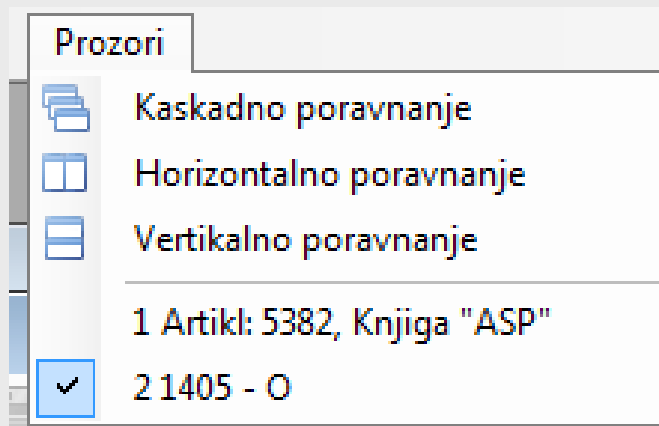
Prikaz aktivnih formi u izborniku

❑ Opcija *Prozori* s listom otvorenih formi i separatorom

- Klik na naziv u listi *Prozori* ili na prozor forme aktualizira odgovarajuću formu
- Zatvaranje forme automatski uklanja njezin naziv iz liste *Prozor*

❑ `Form.MenuStrip.MdiWindowListItem` postavlja se na element izbornika u kojem želimo prikazati aktivne forme/djecu

- `this.menuStrip1.MdiWindowListItem = prozoriToolStripMenuItem; // postavljeno u dizajnu`



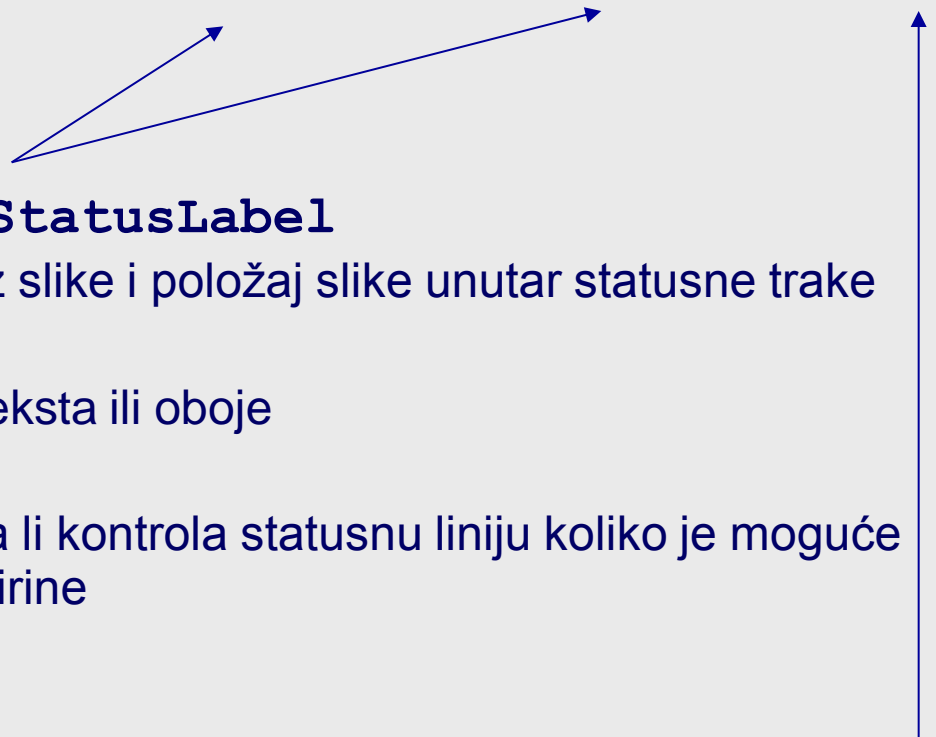
Spajanje izbornika glavne forme i forme djeteta

- ❑ **Glavna forma može udružiti izbornik djeteta u zajednički izbornik**
 - Obavlja se prilikom maksimiziranja forme djeteta unutar glavne forme
 - Svojstvo glavne forme `MainMenuStrip` postavlja se na jedan od izbornika forme s kojim se obavlja udruživanje
 - `this.MainMenuStrip = this.menuStrip1;`

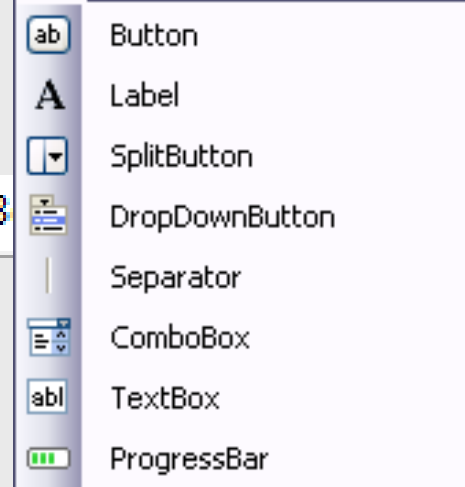
- ❑ **Spajanje pojedinih elemenata izbornika**
 - Svojstvo `MergeAction` elementa izbornika *ToolStripMenuItem*
 - Način na koji se izbornici spajaju
 - `enum MergeAction {Append, Insert, Replace, Remove, MatchOnly}`
 - Svojstvo `MergeIndex` elemenata izbornika *ToolStripMenuItem*
 - Pozicija elementa u zajedničkom izborniku

Statusna traka

- ❑ Kontrola **StatusStrip** – predstavlja statusnu traku (*status bar*)
- ❑ Može sadržavati druge kontrole
 - *ToolStripProgressBar*
 - *ToolStripStatusLabel*
 - Prikaz teksta, slike ili oboje
 - ...
- ❑ Svojstva kontrole **ToolStripStatusLabel**
 - *Image*, *ImageAlign* – prikaz slike i položaj slike unutar statusne trake
 - *Text* – prikaz teksta
 - *DisplayStyle* – prikaz slike/teksta ili oboje
 - *Visible* – zastavica vidljivosti
 - *Spring* – specificira popunjava li kontrola statusnu liniju koliko je moguće (*=true*) ili samo unutar zadane širine
- ❑ Svojstva **StatusStrip**
 - *StatusStrip.SizingGrip* – oznaka razvlačenja u donjem desnom kutu
 - *Form.SizeGripStyle* { *Auto*, *Show*, *Hide* }



Alatna traka



❑ **ToolStrip** – predstavlja alatnu traku (*toolbar*)

- Osnovica za *MenuStrip*, *ContextMenuStrip* i *StatusStrip*

- Svojstva

- `GripStyle { Visible, Hidden }` – vidljivost hvatišta
- `TextDirection – { Horizontal, Vertical90, ... }`
- `Items`: lista kontrola izvedenih iz *ToolStripItem*
 - *ToolStripComboBox*, *ToolStripTextBox*, *ToolStripButton*, ...

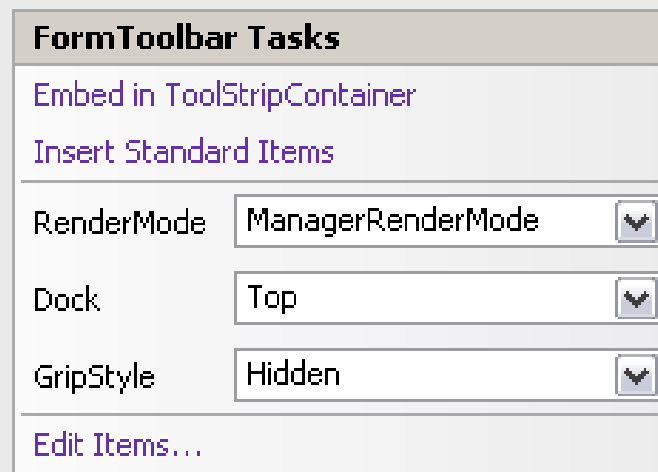
- Događaji:

- `ItemClicked`, ...

❑ **Svojstva *ToolStripButton* kontrole**

- `Enabled`, `Text`, `ToolTipText`, ...

- `DisplayStyle` (enum `ToolStripItemDisplayStyle`)



Događaji pri unosu s tipkovnice

❑ Događaji pridruženi kontrolama i formi, obavljaju se redom

- *KeyDown*
 - *Delegate KeyEventHandler, event arguments KeyEventArgs*
 - pritisnuta je (bilo koja) tipka
- *KeyPress*
 - *Delegate KeyPressEventHandler, event arguments KeyPressEventArgs*
 - pritisnuta je tipka koja predstavlja ASCII znak
- *KeyUp*
 - *Delegate KeyEventHandler, event arguments KeyEventArgs*
 - otpuštena je prethodno pritisnuta tipka

❑ Enumerator *Keys* – nabraja oznake tipki

- `enum Keys { Home, PageUp, ..., Left, Right, A, B, C, F1,...}`

❑ Unos podataka s tipkovnice može biti preduhitren prije nego li ga obradi kontrola u koju se tipka

- formi se postavlja svojstvo `KeyPreview=true`
- primjer: `MainForm_KeyDown`, koji događaj proslijeđuje aktivnoj formi

Argumenti rukovatelja događaja tipkovnice

❑ Svojstva razreda *KeyEventArgs*

- `Alt` – pritisnuta tipka Alt
- `Control` – pritisnuta tipka Control
- `Shift` – pritisnuta tipka Shift
- `Handled` – *bool* oznaka da je događaj obrađen (uklanja "eho")
- `KeyCode` – oznaka tipke kao vrijednost enumeratora `Keys` (npr. `K`)
- `KeyData` – oznaka tipke s informacijom o modifikatoru (npr. `K`, `Control`)
- `KeyValue` – brojučana oznaka tipke (npr. 75)
- `Modifiers` – vrijednost `Keys` za pritisnuti modifikator (npr. `Control`)

❑ Primjer:

- F4 u OS Windows standardno otvara padajuću listu, na što ćemo se osloniti, tj. nećemo sami obrađivati programski, nego postaviti `Handled=false`

❑ Svojstva razreda *KeyPressEventArgs*

- `KeyChar` – pritisnuti osnovni znak (npr. za `SHIFT+K` ima vrijednost veliko `K`)
- `Handled` – *bool* oznaka da je događaj obrađen

Obrada događaja pri unosu s tipkovnice

❑ Primjer: ADO\EF_Firma – DokumentStavkaForm

- Svojstvo *KeyPreview* postaviti na *true*
- Obrada *KeyDown* događaja pozivom odgovarajućih postupaka

```
private async void DokumentStavkaForm_KeyDown(..., EventArgs e) {  
    e.Handled = true;  
    switch (e.KeyCode) {  
        case Keys.Home: First(); break;  
        case Keys.PageUp: Previous(); break;  
        case Keys.PageDown: Next(); break;  
        case Keys.End: Last(); break;  
        case Keys.F1: ShowHelp(); break;  
        case Keys.F2: New(); break;  
        case Keys.F3: Edit(); break;  
        case Keys.F5: Zoom(); break;  
        case Keys.F7: await Delete(); break;  
        case Keys.F9: Cancel(); break;  
        case Keys.F10: await Save(); break;  
        default: e.Handled = false; break;  
    }  
}
```

Ulančavanje procedura

☐ Programska oprema mora slijediti poslovne procese.

- Gdje god je to moguće, treba smanjiti broj postupaka za jedan poslovni proces da korisnici ne bi ponavljali iste akcije.

☐ Kada u sustavu ne postoji potreban vezani podatak korisnik je prisiljen ...

- poništiti do tog trenutka unesene podatke,
- proći kroz nekoliko izbornika do šifarnika, unijeti novu šifru,
- vratiti se na mjesto gdje je šifra bila potrebna,
- ponoviti unos prije poništenih podataka i tek tada ih pohraniti.

☐ Rješenje: na polju za unos vrijednosti stranog ključa omogućiti

- otvaranje “prozora” s listom za pregled i odabir zahtijevanog podatka uz prikaz svih zapisa u odgovarajućoj tablici (*lookup*)
- otvaranje liste za pregled uz prikaz ograničenog skupa zapisa na temelju prethodno postavljenog uvjeta
- aktiviranje funkcije za unos ili izmjenu vezanog podatka
- aktiviranje čitavog modula za obradu vezanih podataka (*zoom*)

Ulančavanje formi stranog ključa

❑ Primjer: ADO\EF_Firma – DokumentStavkaForm (u stanju za uređivanje)

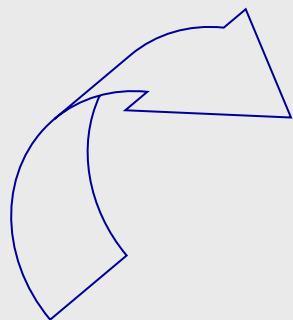
- Pritiskom tipke F5 na polju Partner otvara se forma PartnerForm
- Pritiskom tipke F10 odabrani Partner vraća se u zaglavlje dokumenta

```
private void Zoom()      {
    if (comboBoxPartner.Focused) {
        PartnerForm f = new PartnerForm();
        f.IdPartnera = (int)comboBoxPartner.SelectedValue;
        if (f.ShowDialog() == DialogResult.OK)      {
            // Potrebno osvježiti izvor u slučaju novounesenih partnera
            partnerBindingSource.DataSource =
                context.Partner.AsNoTracking().ToList();

            if (f.IdPartnera != null) {
                ((Dokument)dokumentBindingSource.Current).IdPartnera =
                    f.IdPartnera.Value;
                dokumentBindingSource.ResetCurrentItem();
            }
        }
    }
}
```

Povezivanje formi pri automatizaciji poslovnog procesa

❑ Primjer: EF_Firma – DokumentWizard

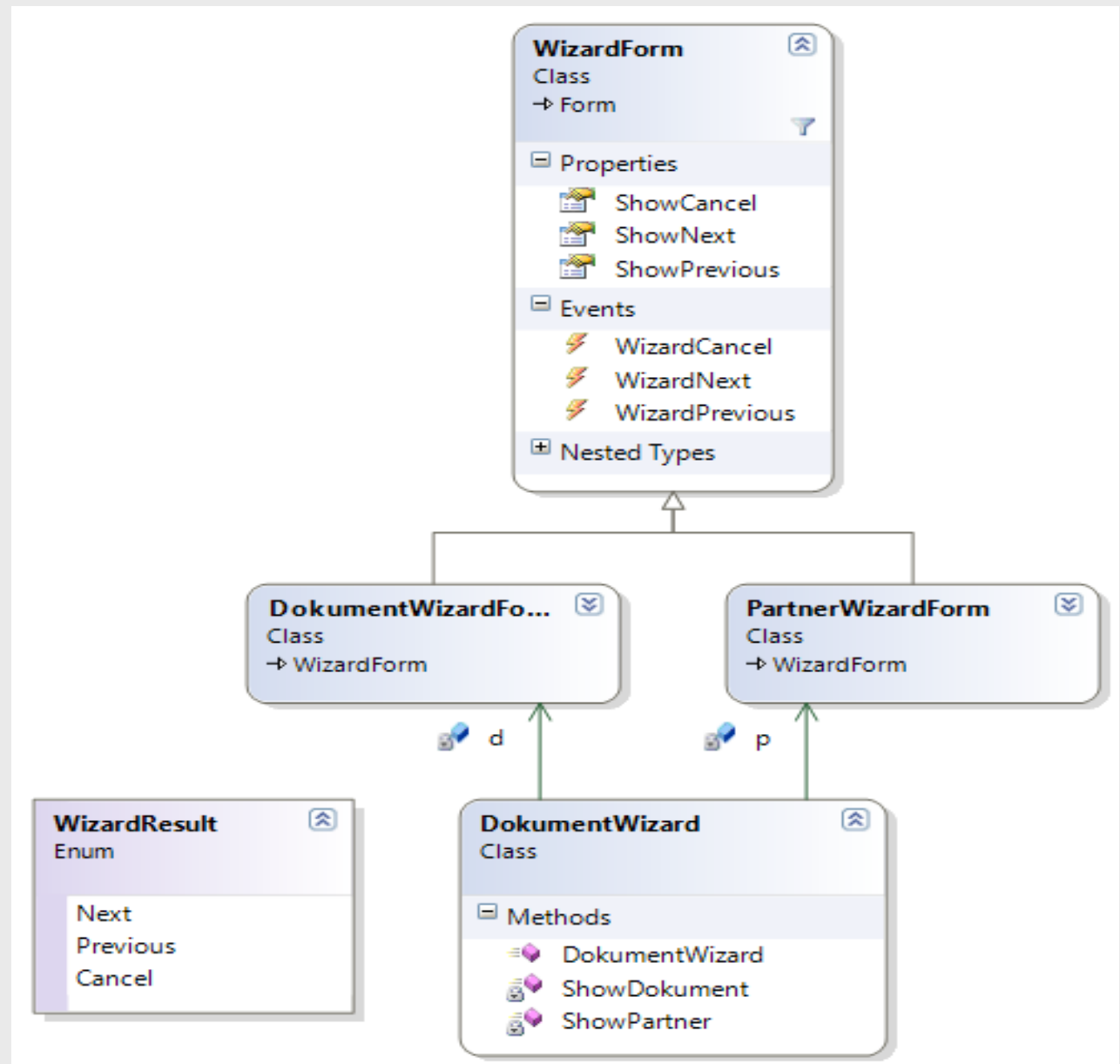


Id dokumenta:	<input type="text"/>	Vrsta:	<input type="text" value="R"/>	Broj:	<input type="text" value="1411"/>	Datum:	<input type="text" value="3. 4.2012."/>	
Partner:	<input type="text" value="976"/>	Ortak, Peti (1234567890123)				Porez:	<input type="text" value="0,00%"/> [0 -1]	
Prethodni:	<input type="text"/>					<input type="button" value="X"/>	Iznos:	<input type="text" value="1445,0000"/>

	Artikl	Količina	Jedinična cijena	Rabat	Ukupna cijena
	PH Leather Belt Case ▼	2	223,0000	0	446,00 kn
▶	Kućno kino DVD/CD/MP3/RECEI... ▼	1	999,0000	0	999,00 kn
*	▼				



Dijagram razreda ulančavanja zaslonskih maski

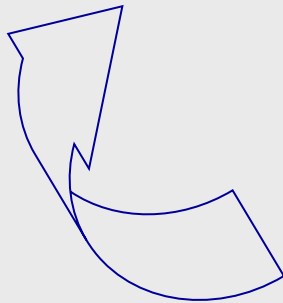
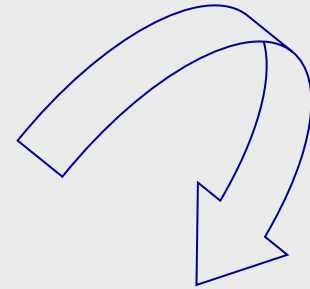


Koraci povezivanja

```
{  
    ...  
    new DokumentWizard()  
}
```

```
public DokumentWizard() {  
    ShowPartner();  
}
```

```
private PartnerWizardForm p;  
private void ShowPartner() {  
    if (p == null)  
        p = new PartnerWizardForm();  
    WizardResult rez = p.ShowWizard();  
    if (rez == WizardResult.Next) {  
        ShowDokument();  
    }  
    ...  
}
```



```
private void ShowDokument() {  
    if (d == null)  
        d = new DokumentWizardForm();  
  
    d.Partner(p.Partner);  
    WizardResult rez = d.ShowWizard();  
    if (rez == WizardResult.Previous) {  
        ShowPartner();  
    }  
}
```

Vlastiti dijalozi

❑ Svojstva forme

- `AcceptButton` – gumb koji će biti aktiviran pritiskom na *Enter*
 - `npr. this.AcceptButton = this.buttonOK;`
- `CancelButton` - gumb koji će biti aktiviran pritiskom na *Escape*

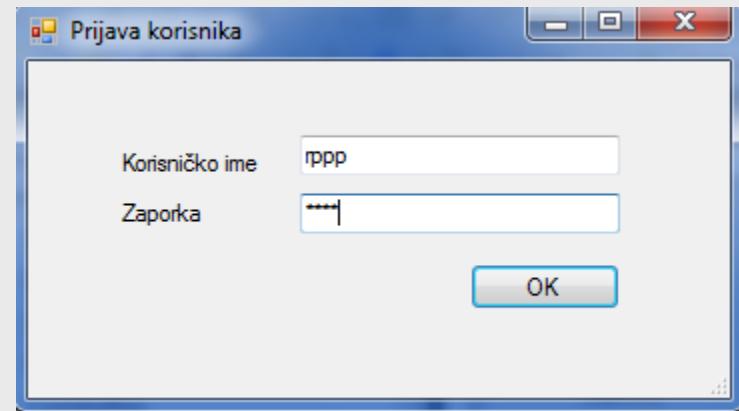
❑ Svojstva gumba

- `DialogResult` – neka od vrijednosti enumeratora `DialogResult`
 - `Abort, Cancel, Ignore, No, None, OK, Retry, Yes`
- na `MODALNOJ` formi, bilo koja postavka izuzev `None` automatski zatvara formu, pa se u obradi događaja može izostaviti `this.Close()`

Dijalog prijave korisnika

❑ Primjer: ADO\EF_Firma – LoginForm

■ `textBoxZaporka.PasswordChar = "*" ;`



```
private void buttonOk_Click(object sender, EventArgs e)
{
    if (ProvjeriKorisnika(textBoxKorisnik.Text,
                           textBoxZaporka.Text))
    {
        DialogResult = DialogResult.OK;
        ...
        Close();
    }
    else
    {
        MessageBox.Show("Krivo korisničko ime ili zaporka.",
                        "Neuspjela autorizacija",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Postavljanje pokazivača miša prilikom dužih operacija

❑ Objedinjeno u razred **StatusBusy**

- Vraćanje prethodnog stanja nakon isteka doseg instance razreda

❑ Primjer: **EF_Firma – StatusBusy**

```
public class StatusBusy : IDisposable{  
    private Cursor c;  
    public StatusBusy(){  
        this.c = Cursor.Current;  
        Cursor.Current = Cursors.WaitCursor;  
    }  
    void IDisposable.Dispose(){  
        Cursor.Current = this.c;  
    }  
}
```

```
using (new StatusBusy())  
{  
    ArtiklForm f = new ArtiklForm();  
    f.MdiParent = this;  
    f.Show();  
}
```

Nadzor rada aplikacije

❑ **Prostor imena `System.Diagnostics`**

- razredi koji omogućavaju ispravljanje pogrešaka (debug) i praćenje izvođenja izvršnog koda (trace)
- `using System.Diagnostics;`

❑ **Razred *EventLog* - omogućuje rukovanje dnevnikom događaja**

- `WriteEntry` – piše zapis u dnevnik događaja
- `WriteEvent` – piše lokalizirani zapis u dnevnik događaja
- `GetEventLogs` – kreira polje sistemskih događaja

❑ **Razred *Debug* - omogućuje ispis na *Output* prozor**

- Metode: `Write`, `WriteLine`, `WriteIf`, `WriteLineIf`
- `npr. Debug.WriteLine ("Ispis na Output prozor");`

❑ **Razred *Trace* - piše na osluškivač traga (trace listener)**

- Metode: `Write`, `WriteLine`, `WriteIf`, `WriteLineIf`

❑ **Razred *TraceListener* – osnovni apstraktni razred osluškivača**

Datoteke

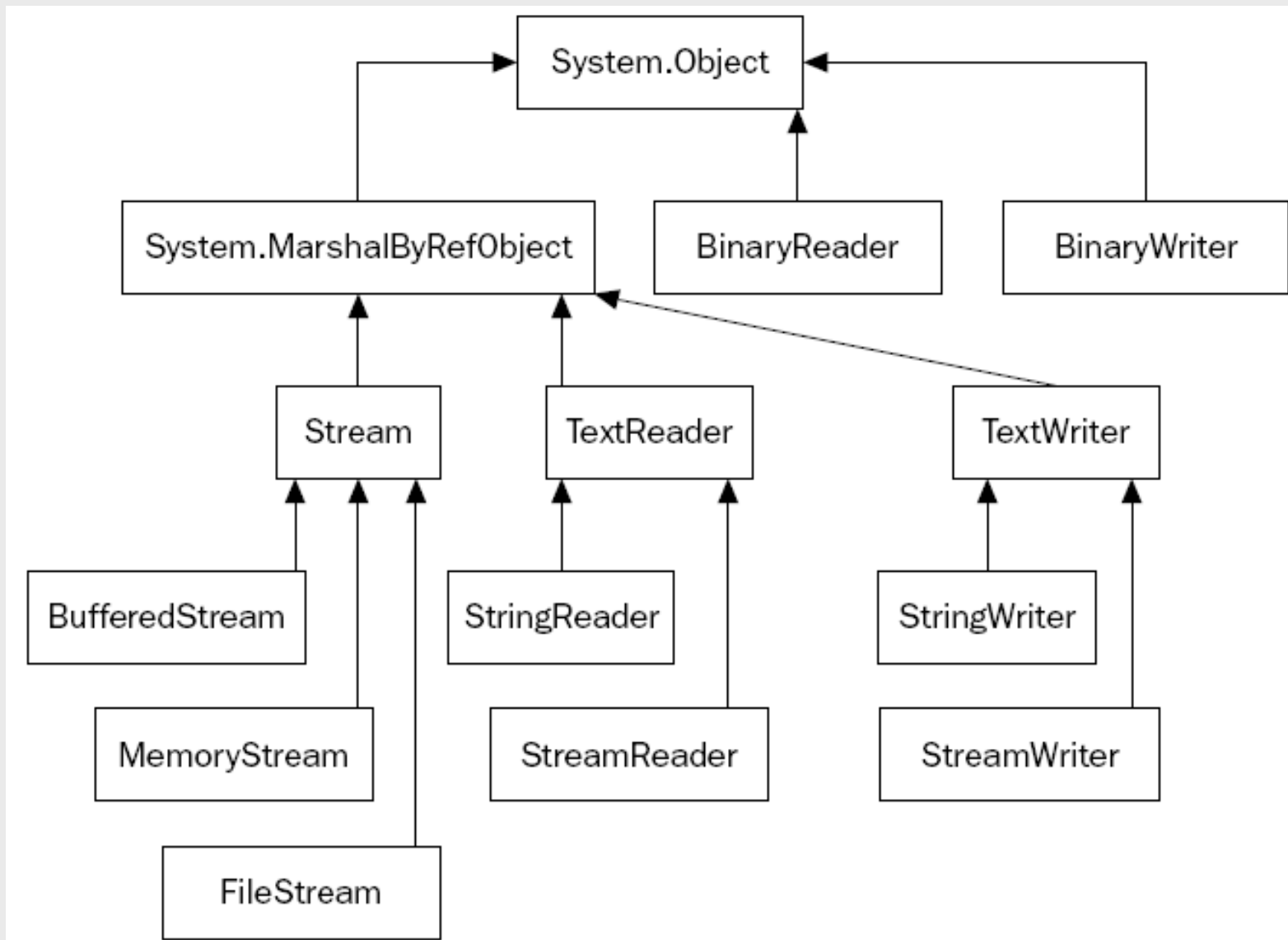
Prostor imena *System.IO*

- ❑ **Ulaz i izlaz (Input/Output tj. I/O) odnose se na postupke čitanja i pisanja podatka putem I/O tokova (*streams*)**
 - Članovi *System.IO* prostora imena koriste se za te postupke

- ❑ **Razredi ovog prostora imena mogu se podijeliti u tri skupine**
 - Razredi za čitanje i pisanje okteta (*bytes*)
 - Razredi za čitanje i pisanje znakova (*character*)
 - Razredi za čitanje i pisanje binarnih podataka

- ❑ **Odluka koji od razreda koristiti ovisi o vrsti podataka**

Hijerarhija razreda



Razred *Stream*

- ❑ Razred *Stream* je apstraktni razred iz kojeg su izvedeni razredi *BufferedStream*, *FileStream* i *MemoryStream* te neki rjeđe korišteni
- ❑ Tok (*stream*) je apstrakcija niza bajtova (npr. datoteka, U/I jedinica, TCP/IP priključnica, međuproceni komunikacijski cjevovod, ...)
- ❑ Na tokove se odnose tri osnovne operacije:
 - Čitanje – prijenos podataka iz toka u neku strukturu podataka
 - Zapisivanje – prijenos podataka iz neke strukture podataka u tok
 - Pozicioniranje – tok može podržavati pozicioniranje po toku (*seeking*), ovisno o svojstvima toka (npr. pri mrežnoj komunikaciji pozicioniranje uglavnom nema smisla)
- ❑ Iako se ne može izravno instancirati objekt razreda *Stream*, postoje postupci koji vraćaju referencu na *Stream* objekt
 - Primjer: `System.Windows.Forms.OpenFileDialog.OpenFile`

Neki od članova razreda `Stream`

❑ **Apstraktna svojstva**

- `bool CanRead` - `true` ukoliko se iz toka može čitati
- `bool CanSeek` - `true` ukoliko tok podržava pomicanje (*seek*)
- `bool CanWrite` - `true` ukoliko se u tok može pisati
- `long Length` - duljina toka u bajtovima
- `long Position` - vraća ili postavlja trenutnu poziciju u toku

❑ **Apstraktni postupci**

- `void Close()` – zatvara tok i oslobađa njime zauzete resurse
- `void Flush()` – zapisuje sadržaj međuspremnika na pripadnu jedinicu
- `int Read(byte[] buffer, int offset, int count)`
 - čita podatke iz toka i pomiče trenutnu poziciju unutar toka
- `void Write(byte[] buffer, int offset, int count)`
 - zapisuje podatke u tok i pomiče trenutnu poziciju unutar toka

❑ **Virtualni postupci**

- `int ReadByte()`
- `void WriteByte(byte value)`

Razred *FileStream*

❑ **FileStream** razred služi za rad s datotekama

❑ **Najčešće korišteni konstruktori:**

```
FileStream(string filename, FileMode mode)
```

```
FileStream(string filename, FileMode mode, FileAccess how)
```

❑ **how** može imati sljedeće vrijednosti:

```
FileAccess.Read, FileAccess.Write, FileAccess.ReadWrite
```

❑ **mode** može imati sljedeće vrijednosti:

`FileMode.Append`

Podaci se nadodaju na kraj datoteke.

`FileMode.Create`

Kreira se nova izlazna datoteka. Postojeća s istim imenom bit će uništena.

`FileMode.CreateNew`

Kreira novu izlaznu datoteku. Datoteka ne smije već postojati.

`FileMode.Open`

Otvora postojeću datoteku.

`FileMode.OpenOrCreate`

Otvora datoteku ako postoji, inače kreira novu.

`FileMode.Truncate`

Otvora postojeću datoteku i odbacuje prethodno postojeći sadržaj (postavlja duljinu na 0).

Razred *FileStream* (nastavak)

❑ Iznimke koje se mogu pojaviti pri pozivu konstruktora `FileStream`:

- | | |
|---|---|
| ■ <code>FileNotFoundException</code> | ako datoteka ne postoji |
| ■ <code>IOException</code> | ako se dogodi pogreška pri otvaranju |
| ■ <code>ArgumentNullException</code> | ako je ime datoteke null |
| ■ <code>ArgumentException</code> | ako je <i>mode</i> parametar neispravan |
| ■ <code>SecurityException</code> | nedostatak prava željenog pristupa |
| ■ <code>DirectoryNotFoundException</code> | zadani direktorij nije valjan |

Primjer kopiranja datoteke

❑ Otvaranje datoteke

```
FileStream fin = new FileStream(args[0], FileMode.Open);  
FileStream fout = new FileStream(args[1], FileMode.Create);
```

❑ Kopiranje datoteka

```
while((i = fin.ReadByte()) != -1) fout.WriteByte((byte)i);
```

❑ Čitanje više bajtova odjednom

```
int bufSize = 10;  
byte[] buf = new byte[bufSize];  
while((i = fin.Read(buf, 0, bufSize)) > 0)  
    fout.Write(buf, 0, i);
```

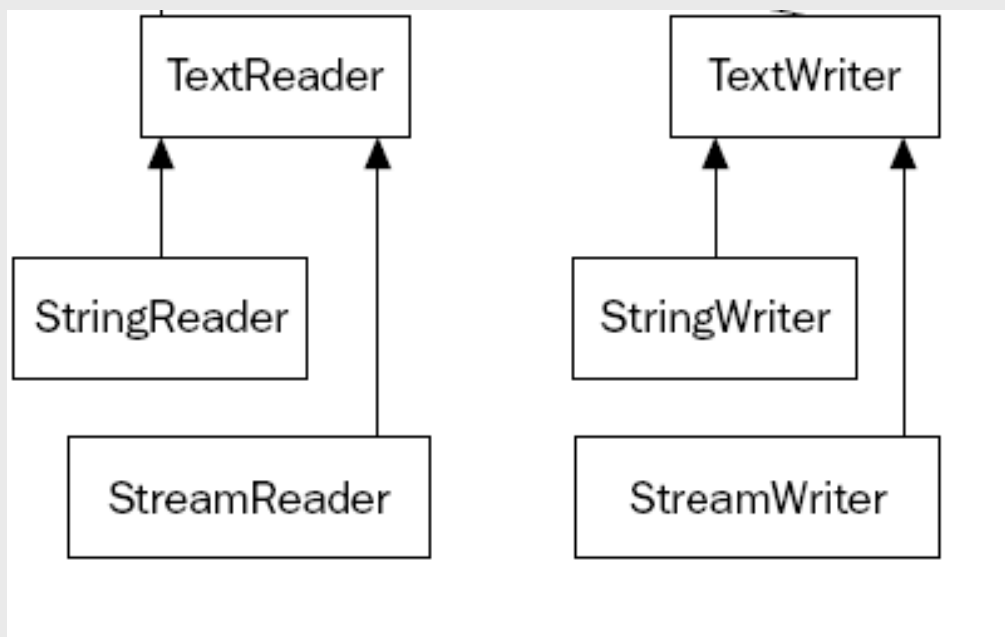
❑ Zatvaranje datoteke

```
fin.Close();  
fout.Close();
```

Znakovni ulaz i izlaz

❑ Razredi za čitanje i zapisivanje slijednih znakova u datoteke, ali i nizove znakova izvedeni su iz razreda:

- `TextReader` – apstraktni razred koji čita slijedni niz znakova
- `TextWriter` – apstraktni razred koji piše slijedni niz znakova



Primjeri *StreamReader* i *StreamWriter*

❑ Prednost pred čitanjem bajtova: izravan rad s *Unicode* znakovima

■ Najčešće korišteni konstruktori:

- `StreamWriter(Stream stream), StreamWriter(String dat), StreamWriter(String dat, bool append)`
- `StreamReader(Stream stream), StreamReader(String dat)`

■ Postupci za čitanje iz datoteke:

- `int Read()` – čita znak, vraća -1 kada nema što čitati
- `int Read(char[] buffer, int offset, int numChars)`
- `int ReadBlock(char[] buffer, int offset, int numChars)`
- `string ReadLine()` // vraća null za dosegnut kraj datoteke
- `string ReadToEnd()`

■ Postupci za pisanje u datoteku:

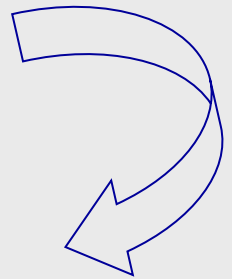
- `void Write(type value) // Write(string), Write(int)`
- `void WriteLine(type value) // koristi ToString + "\n"`

Stream i iznimke

❑ Primjer: Datoteke\Using # BezUsing, TryCatch

```
int a = 7, b = 0;  
StreamWriter sw = new StreamWriter("streamtest.txt");  
sw.WriteLine("Ispis rezultata:");  
sw.WriteLine(string.Format("Drugi red: {0}", a/b));  
sw.Close();
```

```
StreamWriter sw = null;  
try{  
    sw = new StreamWriter("streamtest.txt");  
    sw.WriteLine("Ispis rezultata:");  
    sw.WriteLine(string.Format("Drugi red: {0}", a / b));  
}  
catch (Exception exc){  
    Console.WriteLine("Iznimka..." + exc);  
}  
finally{  
    if (sw != null) sw.Close();  
}
```



Stream i Dispose

❑ **Stream** implementira **IDisposable**, pa je poželjno koristiti *using*

- u ovom slučaju dispose ima isti efekt kao i *Close*
- osigurava zatvaranje datoteke u slučaju iznimke
- olakšava pisanje obrade iznimke (“jednostavniji” kod)

❑ **Primjer:**  **Datoteke \ Using # SUsing**

```
int a = 7, b = 0;
try{
    using(StreamWriter sw = new StreamWriter("streamtest.txt")) {
        sw.WriteLine("Ispis rezultata:");
        sw.WriteLine(string.Format("Drugi red: {0}", a/b));
    }
}
catch (Exception exc){
    Console.WriteLine("Iznimka..." + exc);
}
```

Preddefinirani tokovi

❑ Postoje tri preddefinirana znakovna toka

- `Console.In` (vraća objekt tipa `TextReader`)
- `Console.Out` (vraća objekt tipa `TextWriter`)
- `Console.Error` (vraća objekt tipa `TextWriter`)

❑ Preddefinirani tokovi mogu se preusmjeriti na bilo koji kompatibilni U/I uređaj ili datoteku sljedećim postupcima

- znakovima `<` ili `>` u komandnoj liniji ukoliko to podržava OS
- nekim od postupaka:

```
static void SetIn(TextReader input);  
static void SetOut(TextWriter output);  
static void SetError(TextWriter output);
```

Binarne datoteke

- ❑ Za čitanje i zapisivanje ugrađenih C# tipova podataka u binarnom formatu koristimo razrede `BinaryReader` i `BinaryWriter`
- ❑ Najčešće korišteni konstruktori su:

```
BinaryWriter(Stream outputStream)
```

```
BinaryReader(Stream inputStream)
```
- ❑ Postupak `void Write (tip val)` kao parametar može primiti jedan od sljedećih tipova podataka:

```
sbyte, byte, byte[], bool, short, ushort, int, uint, long, ulong,  
float, double, char, char[], string
```
- ❑ `BinaryReader` definira postupke čitanja za svaki ugrađeni tip
- ❑ `BinaryReader` također definira i sljedeće `Read` postupke:

```
int Read(byte[] buf, int offset, int num)
```

```
int Read(char[] buf, int offset, int num)
```

BinaryReader postupci za čitanje

❑ Postupci za čitanje pojedinog ugrađenog tipa podatka

```
bool ReadBoolean()  
byte ReadByte()  
sbyte ReadSByte()  
byte[] ReadBytes(int num)  
char ReadChar()  
char[] ReadChars(int num)  
double ReadDouble()  
float ReadSingle()  
short ReadInt16()  
int ReadInt32()  
long ReadInt64()  
ushort ReadUInt16()  
uint ReadUInt32()  
ulong ReadUInt64()  
string ReadString() // samo za stringove pisane s BinaryWriter
```

❑ Svi postupci bacaju iznimku `EndOfStreamException` ako je dosegnut kraj datoteke.

Primjer binarne datoteke

```
string artikl; // naziv artikla
int kolicina; // količina na skladištu
double cijena; // jedinična cijena

BinaryWriter bout = new BinaryWriter(new
FileStream("zaliha.dat", FileMode.Create));

// pisanje jednog zapisa
bout.Write("Cekic");
bout.Write(10);
bout.Write(3.95);
...
// čitanje jednog zapisa
artikl = bin.ReadString();
kolicina = bin.ReadInt32();
cijena = bin.ReadDouble();
```

MemoryStream, StringReader, StringWriter

- ❑ Ponekad je korisno čitati ili pisati u neko polje ili string, a ne direktno u neku datoteku na disku.
- ❑ **MemoryStream** je izveden iz razreda **Stream** i koristi polje bajtova za U/I – najčešći konstruktori su:
 - `MemoryStream(byte[] buf)`
 - `MemoryStream()`
- ❑ Ukoliko je umjesto polja bajtova potrebno koristiti `string`, mogu se upotrijebiti **StringReader** i **StringWriter** - konstruktori su:
 - `StringReader(string str)` – implementira `TextReader` koji čita string
 - `StringWriter()` – implementira `TextWriter`, a tekst sprema u automatski kreirani `StringBuilder`
- ❑ Sadržaj pohranjen u **StringWriter** može se dobiti s `ToString()`

Primjer *StringReader*, *StringWriter*

```
// instanciranje pisača
StringWriter strwtr = new StringWriter();

// pisanje "na string"
for(int i=0; i < 10; i++)
    strwtr.WriteLine("Redak: " + i);

// sadržaj pisača
Console.WriteLine (strwtr.ToString());

// instanciranje čitača
StringReader strrdrr =
    new StringReader(strwtr.ToString());
// mogli smo ga i instancirati s podacima
// new StringReader("prvi\ndrugi\ntreci\n");

// čitanje "iz stringa"
string str;
while((str = strrdrr.ReadLine()) != null)
    Console.WriteLine(str);
```


Direktne datoteke

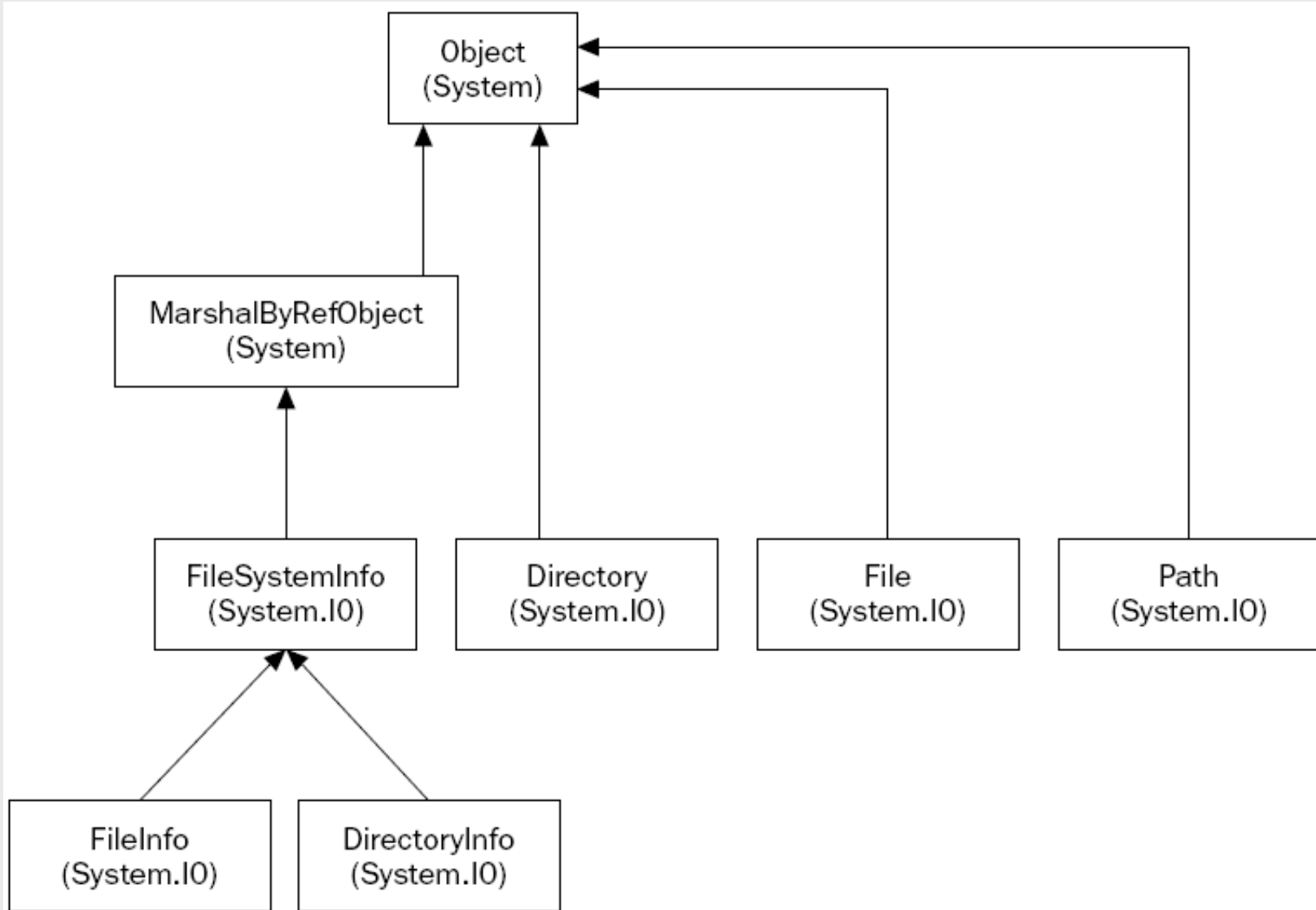
❑ Za pozicioniranje unutar datoteke koristi se postupak:

- `long Seek (long newPos, SeekOrigin origin)`
- `newPos` predstavlja novu poziciju (u bajtovima) pokazivača na datoteku na lokaciju zadanu s `origin`
- `origin` može imati sljedeće vrijednosti:
 - `SeekOrigin.Begin` pozicioniranje od početka datoteke
 - `SeekOrigin.Current` pozicioniranje od trenutne lokacije
 - `SeekOrigin.End` pozicioniranje od kraja datoteke

❑ Primjer:

```
// čitanje s pozicije i iz FileStream f
fileStream.Seek(i, SeekOrigin.Begin);
znak = (char)fileStream.ReadByte();
Console.WriteLine("\n{0}. vrijednost: {1} "
                  , fileStream.Position, znak);
```

Razredi *DirectoryInfo* i *FileInfo*



Razredi *DirectoryInfo* i *FileInfo*

❑ Apstraktni razred *System.IO.FileSystemInfo*

- Bazni razred za razrede `DirectoryInfo` i `FileInfo`
- *FileSystemInfo* sadrži i sljedeće članove:

<code>Attributes</code>	Vraća ili postavlja <i>FileAttributes</i> kolekciju (enumeraciju) s atributima datoteke (skrivena, normalna, jedinica, ...)
<code>CreationTime</code>	Vraća vrijeme stvaranja datoteke
<code>Exists</code>	Postoji li zadana datoteka
<code>Extension</code>	Vraća string koji predstavlja ekstenziju datoteke
<code>FullName</code>	Potpuni naziv (apsolutni) datoteke
<code>LastAccessTime</code>	Vraća ili postavlja vrijeme zadnjeg pristupa datoteci
<code>LastWriteTime</code>	Vraća ili postavlja vrijeme zadnjeg zapisivanja u datoteku
<code>Delete</code>	Briše datoteku
<code>Refresh</code>	Osvježava stanje datoteke

❑ Napomena: direktorij je također "datoteka"

❑ *FileAttributes* enumeracija ima (između ostalog) sljedeće članove:

- `Archive`, `Directory`, `Hidden`, `Normal`, `ReadOnly`, `System`

Razred *DirectoryInfo*

- ❑ Razred `DirectoryInfo` - stvaranje, premještanje (preimenovanje) i iteraciju (enumeraciju) kroz (pod)direktorije
- ❑ Neki članovi razreda `DirectoryInfo`

Parent	Vraća nad-direktorij direktorija
Root	Vraća osnovni (root) dio direktorija (u stvari naziv diska)
Create	Stvara direktorij
CreateSubDirectory	Stvara poddirektorij (ili više poddirektorija)
GetDirectories	Vraća polje objekata tipa <code>DirectoryInfo</code> koje odgovara poddirektorijima
GetFiles	Vraća polje objekata tipa <code>FileInfo</code> koje odgovara datotekama u tom direktoriju
MoveTo	Premješta direktorij predstavljen instancom objekta na drugo mjesto

❑ Primjer uporabe Datoteke\ Using

```
DirectoryInfo dir1 = new DirectoryInfo(@"C:\WINDOWS");  
Console.WriteLine("Full Name is : {0}", dir1.FullName);  
Console.WriteLine("Created: {0}",  
    dir1.CreationTime.ToString("d.M.yyyy. HH:mm:ss"));
```


Svojstva datoteka

❏ Primjer: Datoteke\Using # Info

- Prikaz informacija o odabranoj datoteci

```
FileInfo datoteka = new FileInfo("naziv datoteke");  
...  
string detalji =  
"Puno ime datoteke: " + datoteka.FullName + "\n"  
+ "Ime direktorija: " + datoteka.DirectoryName + "\n"  
+ "Ekstenzija: " + datoteka.Extension + "\n"  
+ "Veličina: " + datoteka.Length + " B\n"  
+ "Vrijeme stvaranja: " + datoteka.CreationTime + "\n"  
+ "Vrijeme zadnje izmjene: " + datoteka.LastWriteTime + "\n"  
+ "Vrijeme zadnjeg pristupanja: " + datoteka.LastAccessTime;
```

Serijalizacija

- ❑ Postupak kojim se vrijednosti objekta pospremaju u neki tok.
- ❑ Deserijalizacijom se na osnovu spremljenih podataka objekt naknadno rekonstruira.
- ❑ Binarna serijalizacija
 - Iznad razreda dopisati [Serializable]
 - uključuje sve varijable i svojstva osim varijabli iznad kojih je atribut [NonSerialized]
- ❑ Xml serijalizacija
 - pohranjuje samo javne varijable i javna svojstva koja imaju i get i set i nemaju atribut [XmlIgnore]
 - Neki tipovi se ne mogu serijalizirati (npr. Hashtable)
 - Mora postojati prazni konstruktor (koristi se prilikom deserijalizacije)
- ❑ Primjer:  Datoteke\Serijalizacija

```
Podatak p = new Podatak("A", "B", "C", "D", "E", "F", "G");  
BinaryFormatter serializer = new BinaryFormatter();  
serializer.Serialize(stream, p);  
...  
Podatak podatak = (Podatak) serializer.Deserialize(stream);
```

Reference

- ❑ **Scott W. Ambler:User Interface Design Tips, Techniques, and Principles**
 - <http://www.ambysoft.com/essays/userInterfaceDesign.html>
- ❑ **Papers and Essays by Jakob Nielsen**
 - <http://www.nngroup.com/articles/>
- ❑ **Human-Computer Interfaces**
 - <http://www.cs.umd.edu/hcil/>
- ❑ **Windows User Experience Interaction Guidelines**
 - <http://msdn.microsoft.com/en-us/library/aa511258.aspx>