

# **Datoteke, prijenos podataka**

---

**10/13**

# Rad s *Excel* datotekama

## ❑ MS *Excel* datoteke možemo čitati/pisati koristeći *OleDb Provider*

- Data Source navodi puni put do datoteke
- Extended Properties označava dodatna svojstva:
  - Excel 8.0 označava priključak na *Excel* datoteku verzije 8.0 – nužno ga je navesti za rad s *Excel* datotekama
  - HDR=Yes označava da se u prvom retku nalaze nazivi stupaca tablice

```
"Provider=Microsoft.Jet.OLEDB.4.0;  
Data Source=C:\\Projects\\podaci.xls;  
Extended Properties='Excel 8.0; HDR=Yes'"
```

## ❑ Čitanje podataka iz lista (*sheet*)

- Naziv lista navodi se u uglatim zagradama

```
OleDbCommand datotekaComm = new OleDbCommand(  
    "SELECT * FROM [" + nazivLista + "]", datotekaConn);
```

- npr. "SELECT \* FROM [Artikl\$]"
- \$ u nazivu lista je standardna interna oznaka Excel-a

# Rad s datotekama s ograničivačem

## ❑ Tekstovne datoteke s ograničivačem (delimiter, separator)

- ograničivač - znak koji se ne smatra podatkom (zarez, razmak, tabulator, ...)
- CSV datoteke (*Comma Separated Values*) – ograničivač je zarez


## ❑ Čitamo koristeći *OleDb Provider*

- DataSource navodi mapu (folder) datoteke
- Text označava da se spajamo na tekstovnu datoteku
- FMT=Delimited označava da se radi o formatu datoteke s ograničivačima
  - Ograničivač je standardno zarez (,)

```
"Provider=Microsoft.Jet.OLEDB.4.0;  
Data Source=C:\\Projects;  
Extended Properties='Text; FMT=Delimited; HDR=Yes'"
```

```
OleDbCommand datotekaComm = new OleDbCommand(  
    "SELECT * FROM [" + nazivDatoteke + "]", datotekaConn);
```

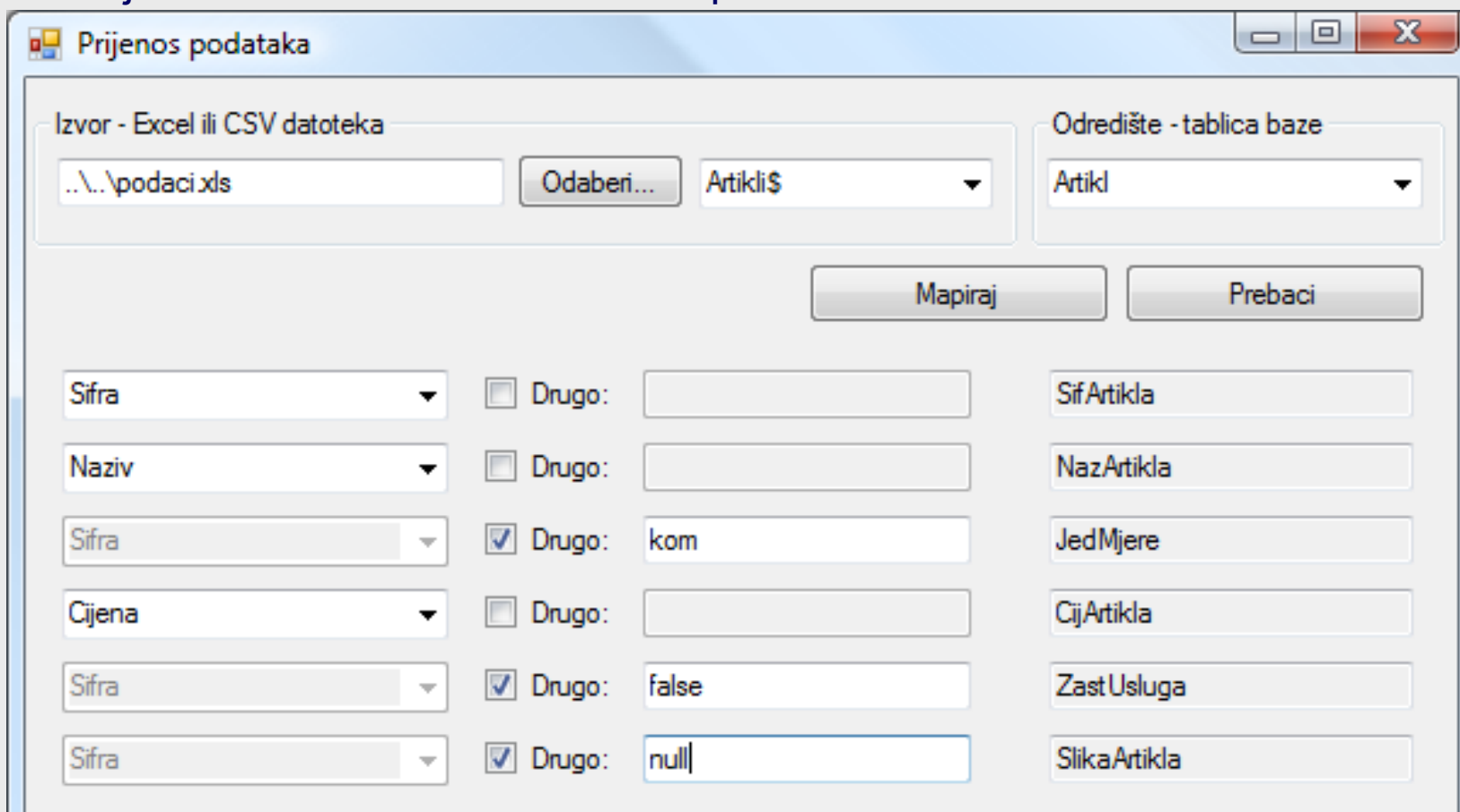
## ❑ *Schema.ini* - definira neki drugi ograničivač umjesto standardnog

- mora se nalaziti u direktoriju datoteke s podacima
- Primjer:  Datoteke \ Import \ schema.ini, redak FMT=Delimited(;;)

# Primjer prijenosa podataka iz Excel/CSV datoteke

## ❑ Primjer: Datoteke \ Import

- Čitanje XLS/CSV datoteke (u prvom retku su nazivi stupaca)
- Mapiranje stupaca XLS/CSV datoteke i stupaca tablice baze podataka
- Prijenos u odabranu tablicu baze podataka



**Prijenos podataka**

Izvor - Excel ili CSV datoteka

..\..\podaci.xls

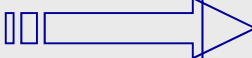
Artikli\$

Odredište - tablica baze

Artikl

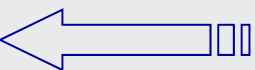
Sifra	<input type="checkbox"/>	Drugo:		SifArtikla
Naziv	<input type="checkbox"/>	Drugo:		NazArtikla
Sifra	<input checked="" type="checkbox"/>	Drugo:	kom	JedMjere
Cijena	<input type="checkbox"/>	Drugo:		CijArtikla
Sifra	<input checked="" type="checkbox"/>	Drugo:	false	ZastUsluga
Sifra	<input checked="" type="checkbox"/>	Drugo:	null	SlikaArtikla

# Prilagodba ovisno o tipu datoteke ...

```
private void PostaviListove()  
{  
    string ekst = Ekstenzija(datoteka);  
    if (ekst == ".xls")  
    {  
        comboBoxList.Enabled = true;  
        comboBoxList.DataSource = DohvatiListoveDatoteke();   
        nazivLista = "[" + comboBoxList.Text + "]"; // [Artikl$]  
    }  
    if (ekst == ".csv" || ekst == ".txt")  
    {  
        comboBoxList.Enabled = false;  
        comboBoxList.DataSource = null;  
        nazivLista = datoteka; // podaci.csv  
    }  
}
```

# Dohvat naziva listova Excel datoteke

```
private List<string> DohvatiListoveDatoteke()  
{  
    List<string> listovi = new List<string>();  
    try  
    {  
        OtvoriDatotekaKonekciju();  
  
        // vraća tablicu s nazivima listova Excelsa  
        DataTable tablicaShema = datotekaConn.GetOleDbSchemaTable(  
                                                    OleDbSchemaGuid.Tables, null);  
        foreach (DataRow row in tablicaShema.Rows)  
        {  
            listovi.Add(row["TABLE_NAME"].ToString());  
        }  
    }  
    ...  
    return listovi;  
}
```



```
comboBoxList.DataSource = DohvatiListoveDatoteke();
```

# Dohvat naziva tablica u bazi podataka

```
private List<string> DohvatiTabliceBaze()  
{  
    List<string> tablice = new List<string>();  
  
    try {  
        OtvoriSqlKonekciju();  
  
        SqlCommand dohvatiTablice =  
            new SqlCommand("SELECT table_name  
                            FROM INFORMATION_SCHEMA.Tables  
                            WHERE table_type = 'BASE TABLE'", sqlConn);  
        SqlDataReader tabliceReader = dohvatiTablice.ExecuteReader();  
  
        while (tabliceReader.Read())  
        {  
            tablice.Add(tabliceReader[0].ToString());  
        }  
        ...  
        return tablice;  
    }  
}
```

```
comboBoxTablice.DataSource = DohvatiTabliceBaze();
```

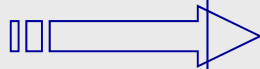
# Kontrole za mapiranje

```
private void PostaviMapiranje() {  
...  
    // dodajemo user kontrole za svaki stupac tablice u bazi podataka  
    int y = 110;  
    for (int i = 0; i < dt.Columns.Count; i++)  
    {  
        UserControlMapiranje mapiranje = new UserControlMapiranje();  
  
        mapiranje.ComboBindingContext = new BindingContext();  
...  
        mapiranje.Name = "kontrolaMapiranje";  
        mapiranje.Location = new Point(0, y);  
        y += 30; // prored  
  
        // ako je stupac auto increment (identity), ne dopustamo editiranje  
        if (dt.Columns[i].AutoIncrement) mapiranje.Onemoguci();  
  
        this.Controls.Add(mapiranje);  
    }  
    buttonPrebaci.Enabled = true;  
}
```



# Prijenos podataka iz datoteke u tablicu

```
private void PrebaciPodatke() {  
  
    while (datotekaReader.Read()) // za svaki zapis datoteke ...  
    {  
        // ... prikupimo vrijednosti  
        stupciMapirani = DohvatiVrijednosti(datotekaReader);  
  
        // sastavljamo select upit (po primarnom kljucu)  
        adapter.SelectCommand.CommandText =  
            SelectPoKljucu(stupciMapirani);  
  
        dt.Clear();  
        adapter.Fill(dt); // punimo DataTable selektiranim retkom  
  
        if (dt.Rows.Count == 0) // ako zapis ne postoji u BP  
        {  
            DataTableInsert(stupciMapirani); // dodajemo novi  
        }  
        else // inače ažuriramo postojeći  
        {  
            DataTableUpdate(stupciMapirani);  
        }  
  
        // prosljedimo promjene u bazu podataka  
        adapter.Update(dt);  
    }  
}
```



# Formiranje zapisa

```
private List<string> DohvatiVrijednosti(OleDbDataReader reader)
{
    List<string> vrijednosti = new List<string>();
    try
    {
        Control[] kontrole =
            this.Controls.Find("kontrolaMapiranje", false);
        foreach (Control kontrola in kontrole)
        {
            UserControlMapiranje k = (UserControlMapiranje)kontrola;
            if (k.OstaloChecked)
            {
                vrijednosti.Add(k.OstaloText);
            }
            else
            {
                vrijednosti.Add(reader[k.ComboText].ToString());
            }
        }
    }
    ...
    return vrijednosti;
}
```

# Dodavanje/ažuriranje zapisa

**// umetanje novog zapisa**

```
private void DataTableInsert(List<string> vrijednosti)
```

...

```
    DataRow dr = dt.NewRow();
    for (int i = 0; i < vrijednosti.Count; i++) {
        if (vrijednosti[i] == "null") {
            dr[i] = DBNull.Value;
        } else {
            dr[i] = vrijednosti[i];
        }
    }
    dt.Rows.Add(dr);
```

**// ažuriranje vrijednosti zapisa dohvaćenog po primarnom ključu**

```
private void DataTableUpdate(List<string> vrijednosti)
```

...

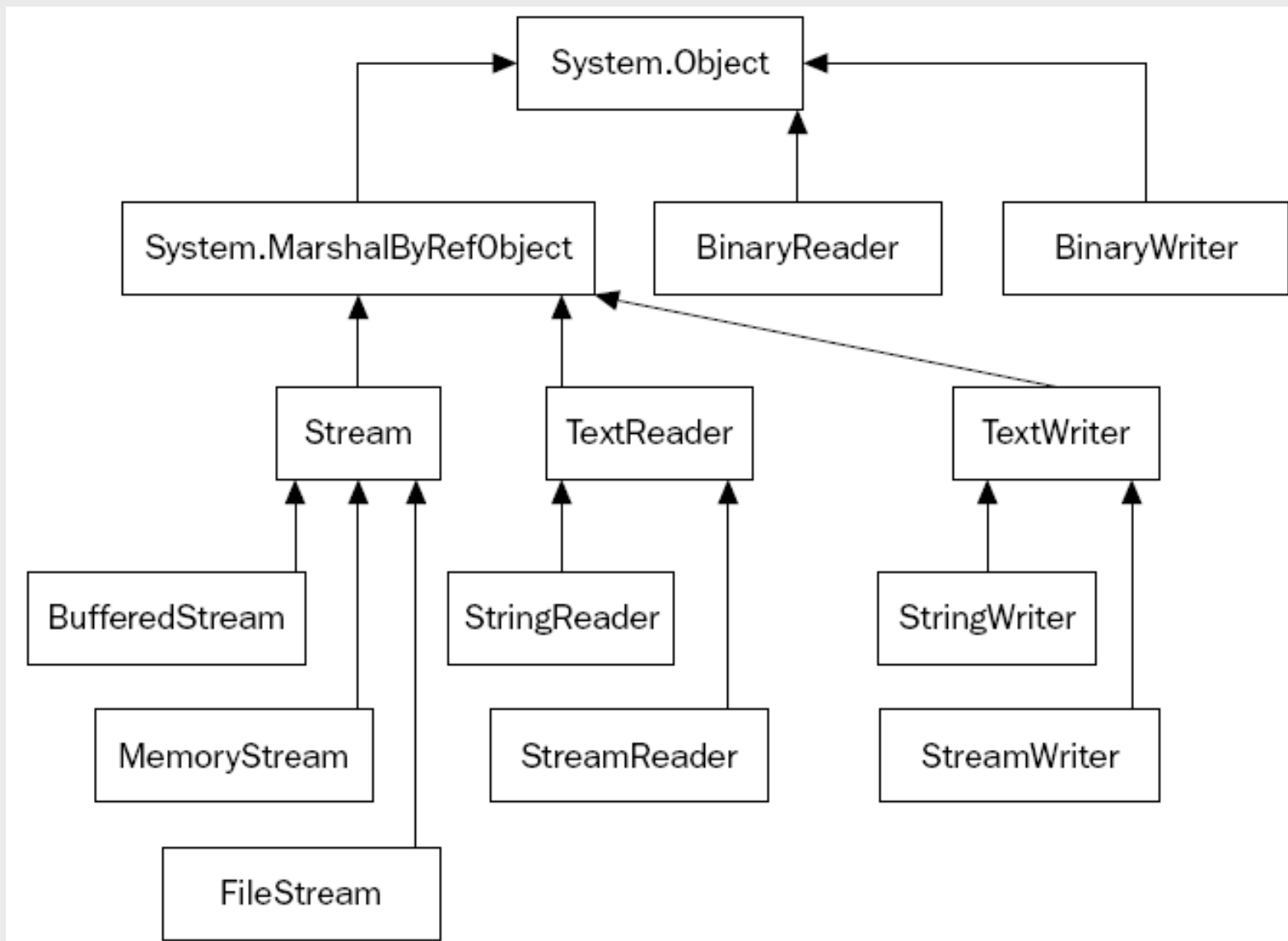
```
    for (int i = 0; i < vrijednosti.Count; i++) {
        if (!dt.Columns[i].AutoIncrement) {
            if (vrijednosti[i] == "null") {
                dt.Rows[0][i] = DBNull.Value;
            } else {
                dt.Rows[0][i] = vrijednosti[i];
            }
        }
    }
```

...

# 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 datoteka

## ❑ 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);
```

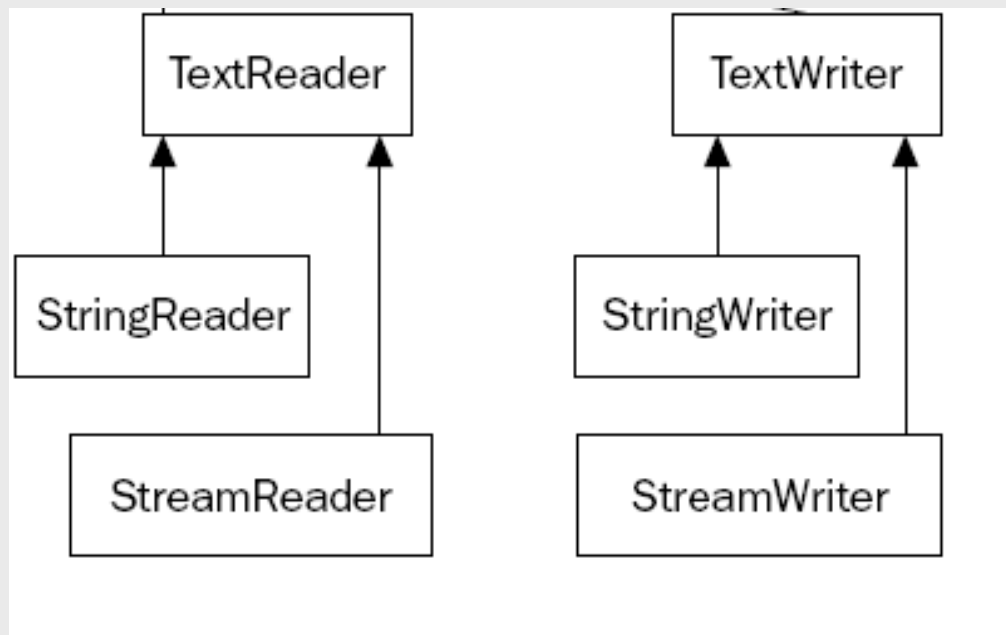
## ❑ Zatvaranje datoteke

```
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)`
- `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:

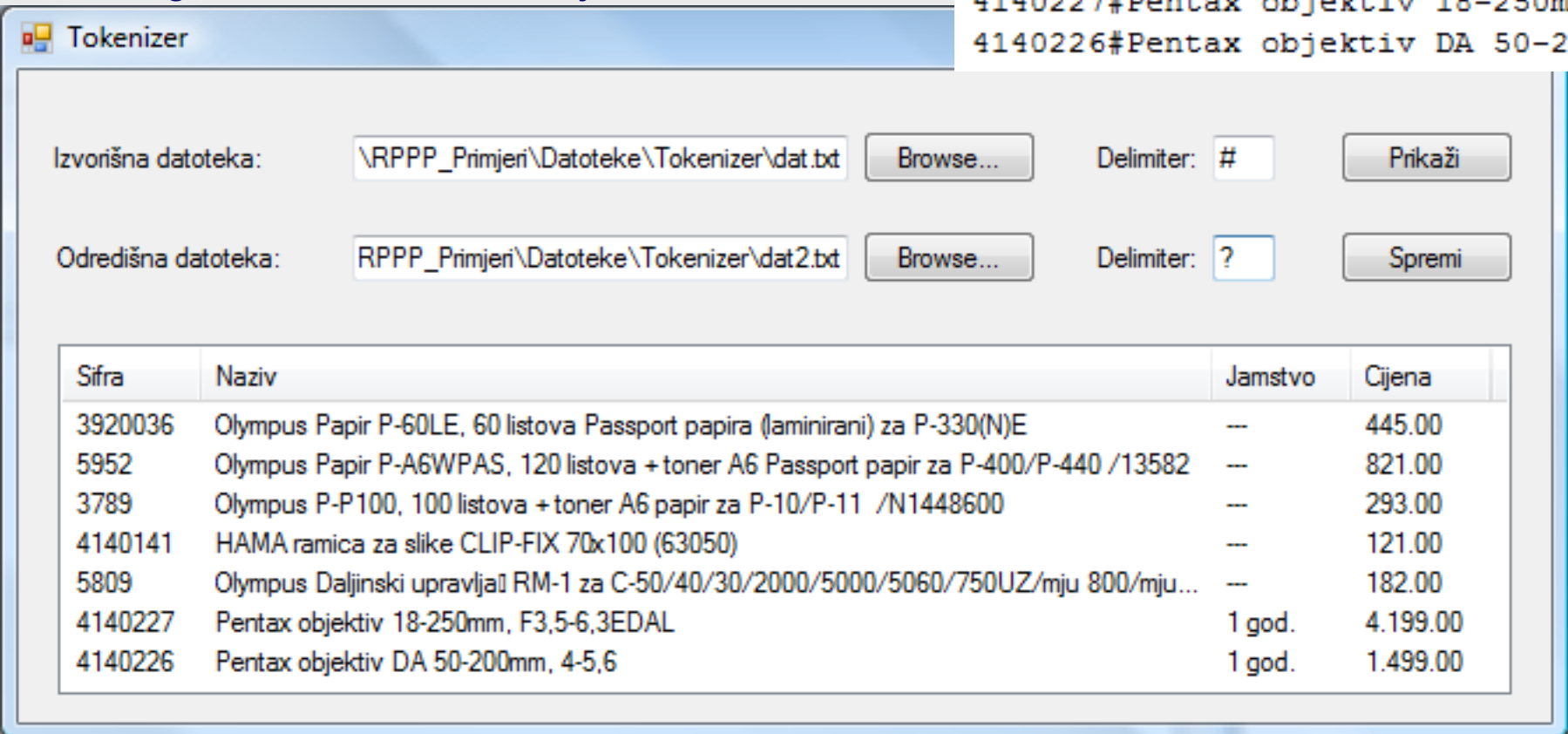
- `void Write( type value ) // Write(string), Write(int)`
- `void WriteLine(type value ) // koristi ToString + "\n"`

# Primjer datoteke s ograničivačem

## ❑ Primjer: Datoteke\Tokenizer

- Učitavanje vrijednosti iz datoteke s ograničivačem, korištenjem *StreamReader*
- Spremanje u novu datoteku s novim ograničivačem, korištenjem *StreamWriter*

```
Sifra#Naziv#Jamstvo#Cijena
3920036#Olympus Papir P-60LE, 60
5952#Olympus Papir P-A6WPAS, 120
3789#Olympus P-P100, 100 listova
4140141#HAMA ramica za slike CLI
5809#Olympus Daljinski upravljač
4140227#Pentax objektiv 18-250mm
4140226#Pentax objektiv DA 50-20
```



Izvorišna datoteka: \RPPP\_Primjeri\Datoteke\Tokenizer\dat.txt Browse... Delimiter: # Prikaži

Odredišna datoteka: RPPP\_Primjeri\Datoteke\Tokenizer\dat2.txt Browse... Delimiter: ? Spremi

Sifra	Naziv	Jamstvo	Cijena
3920036	Olympus Papir P-60LE, 60 listova Passport papira (laminirani) za P-330(N)E	---	445.00
5952	Olympus Papir P-A6WPAS, 120 listova + toner A6 Passport papir za P-400/P-440 /13582	---	821.00
3789	Olympus P-P100, 100 listova + toner A6 papir za P-10/P-11 /N1448600	---	293.00
4140141	HAMA ramica za slike CLIP-FIX 70x100 (63050)	---	121.00
5809	Olympus Daljinski upravljač RM-1 za C-50/40/30/2000/5000/5060/750UZ/mju 800/mju...	---	182.00
4140227	Pentax objektiv 18-250mm, F3,5-6,3EDAL	1 god.	4.199.00
4140226	Pentax objektiv DA 50-200mm, 4-5,6	1 god.	1.499.00

# Primjer datoteke s ograničivačem

## ❑ Čitanje datoteke i odvajanje vrijednosti

```
StreamReader reader = new StreamReader(nazivDatoteke);  
while ((redak = reader.ReadLine()) != null)  
{  
    string lista = redak.Split(new char[] { delimiter });  
    for (int i = 0; i < lista.Length; i++) ...  
}  
reader.Close();
```

## ❑ Pisanje u novu datoteku s novim ograničivačem

```
StreamWriter writer = new StreamWriter(nazivDatoteke);  
for (int i = 0; i < listViewPodaci.Items.Count; i++)  
{  
    string redak = listViewPodaci.Items[i].Text;  
    for (int j = 1; j < listViewPodaci.Items[i].SubItems.Count; j++)  
        redak += delimiter + listViewPodaci.Items[i].SubItems[j].Text;  
    writer.WriteLine(redak);  
}  
writer.Close();
```

# 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 - jedan od konstruktora je:
  - `MemoryStream(byte[] buf)`
- ❑ 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

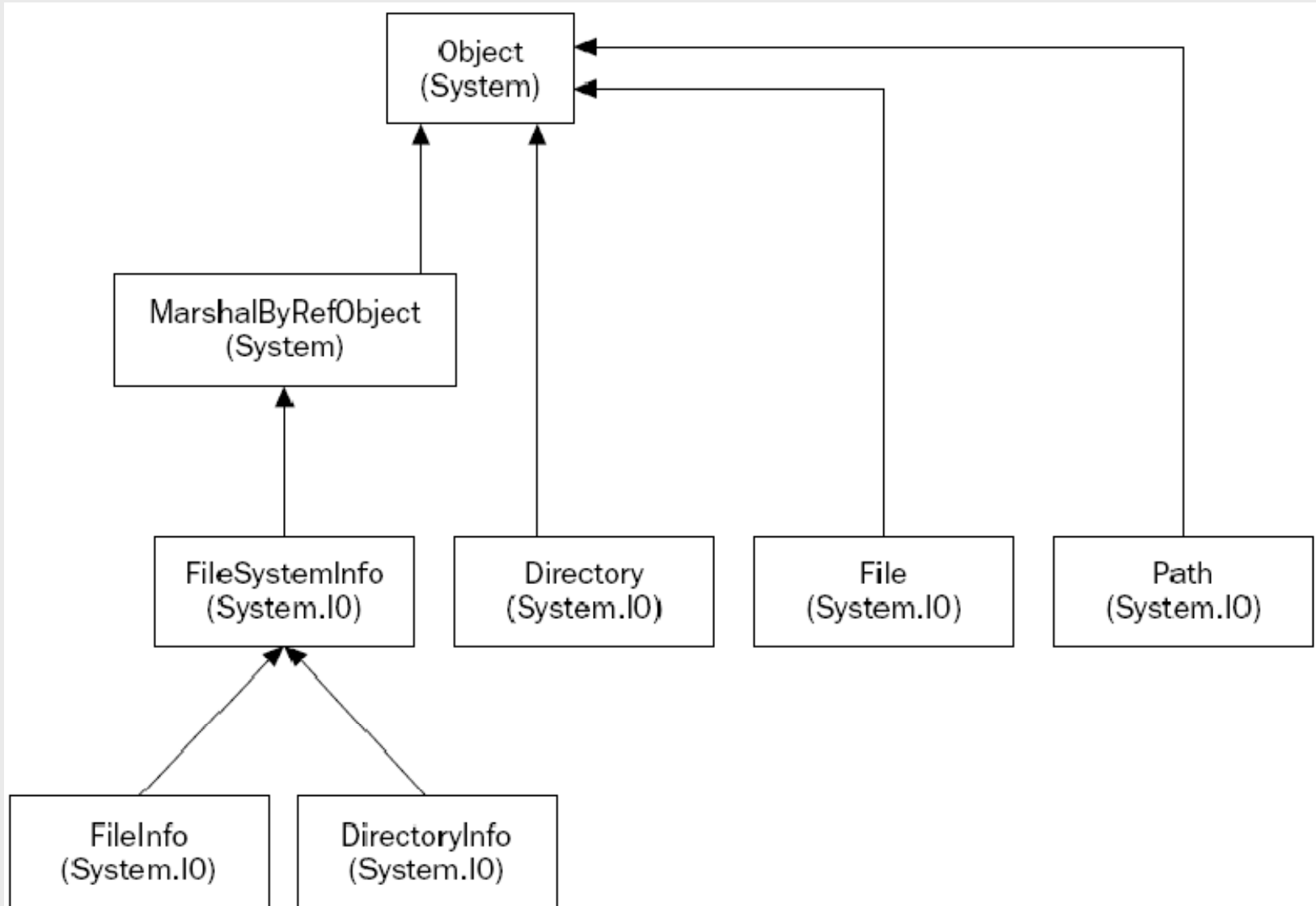
## ❑ Za pozicioniranje unutar datoteke koristi se postupak:

- `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*
- *FileInfo* 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

```
DirectoryInfo dir1 = new DirectoryInfo(@"C:\WINDOWS");  
Console.WriteLine("Full Name is : {0}", dir1.FullName);  
Console.WriteLine("Attributes are : {0}",  
    dir1.Attributes.ToString());
```



# Izbornik zavisan o kontekstu

## ❑ **ContextMenuStrip**

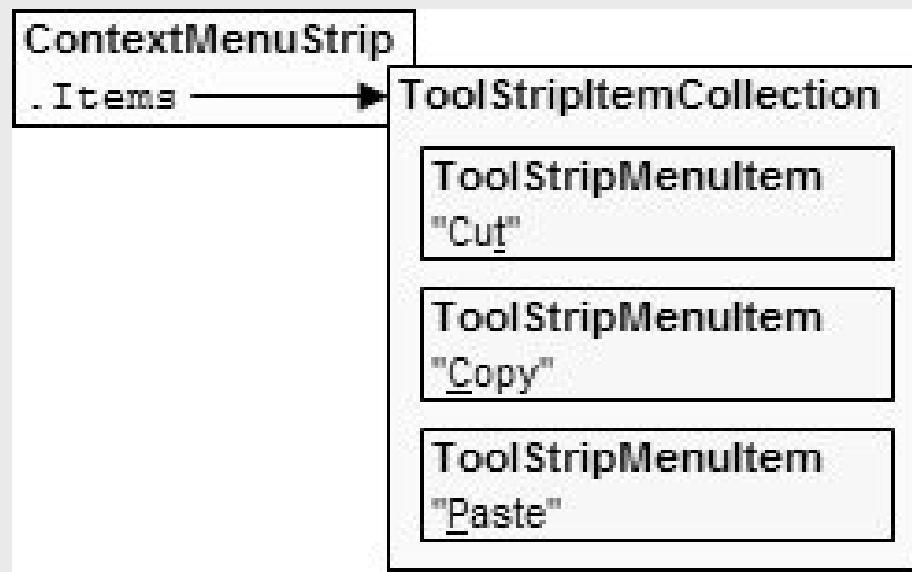
- izbornik zavisan o kontekstu (brzi izbornik, skočni izbornik, izbornik prečice)
- sadrži kolekciju *Items* tipa *ToolStripItemCollection*
- može imati hijerarhiju opcija ali nema horizontalnu početnu komponentu

## ❑ **Forma / kontrola ima najviše jedan izbornik zavisan o kontekstu**

- *ContextMenuStrip* svojstvo forme / kontrole

## ❑ **Primjer: GUI\ListaDatoteka**

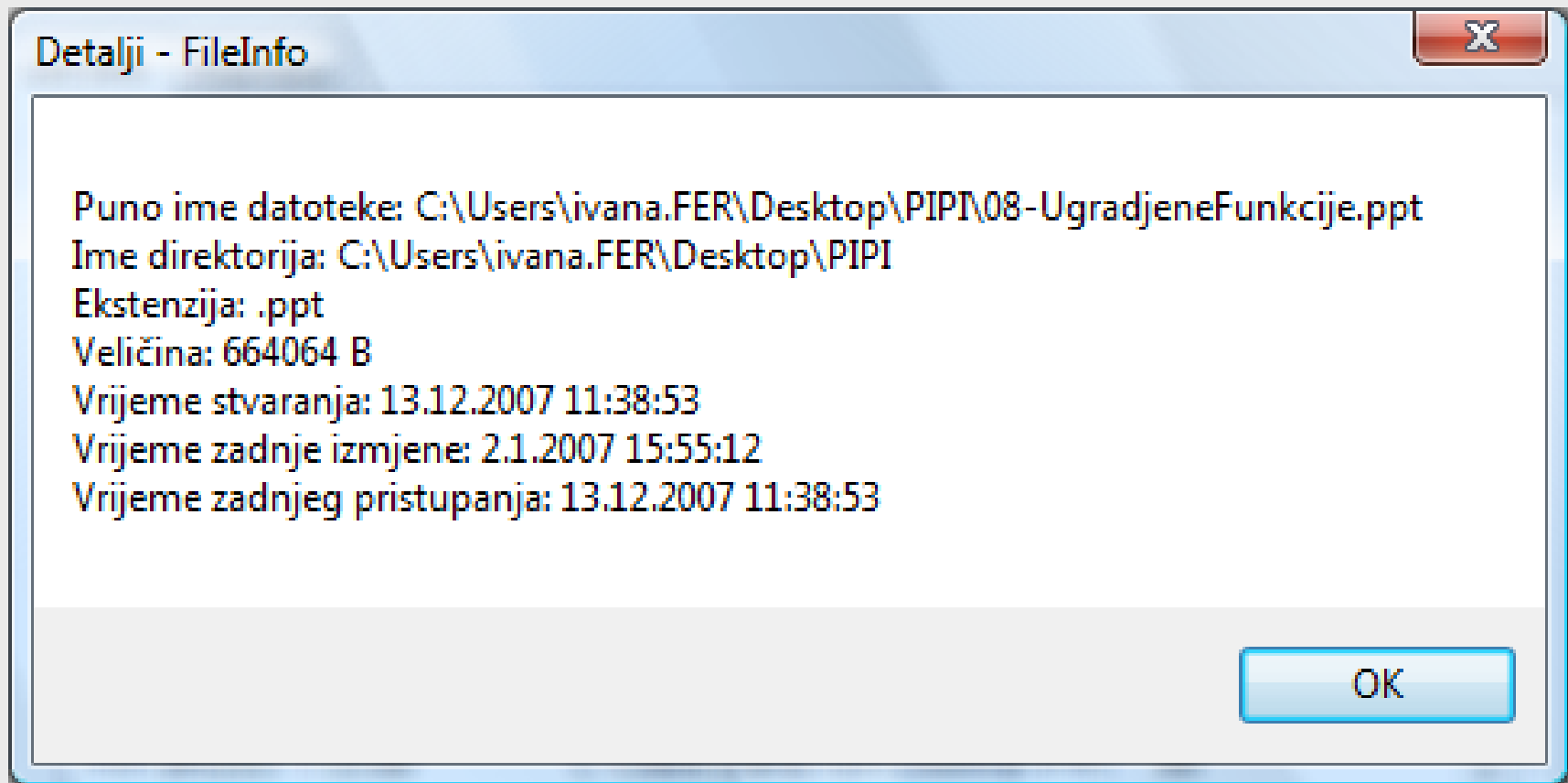
- `listView.ContextMenuStrip  
= this.contextMenuStrip`



# Primjer izlista datoteka

## ❑ Primjer: GUIListaDatoteka

- Prikaz datoteka/direktorija u *ListView* i *TreeView* kontroli
- Desni klik na element u *ListView* – detalji o odabranom elementu



# Rekurzivni prikaz datoteka

## ❏ Primjer: 📁 GUIListaDatoteka – Form1.cs

```
private void PrikaziDokumenteListView(string imeDirektorija)
{
    DirectoryInfo directoryInfo =
        new DirectoryInfo(imeDirektorija);
    FileSystemInfo[] listFiles =
        directoryInfo.GetFileSystemInfos();
    foreach (FileSystemInfo fileSystemInfo in listFiles)
    {
        if (fileSystemInfo is FileInfo)        {
            ListViewItem listItem =
                new ListViewItem(fileSystemInfo.Name);
            ...
        } else if (fileSystemInfo is DirectoryInfo)    {
            PrikaziDokumenteListView(fileSystemInfo.FullName);
        }
    }
}
```

# Svojstva datoteka

## ❑ Prikaz informacija o odabranoj datoteci

```
FileInfo datoteka = new FileInfo(oznaceniItem.SubItems[1].Text
    + "\\\" + oznaceniItem.Text);

...
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
```

## ❑ Implementacija desnog klika na element liste

- Primjer: `listView1_MouseDown`
- dohvat koordinata, `contextMenuStrip.Show`



# Zadaci za vježbu

- ❑ **Napisati aplikaciju koja binarno uspoređuje dvije datoteke**
  
- ❑ **Napisati aplikaciju za grupni (*batch*) uvoz slike**
  - Id artikla i naziv slike (put do slike) zadani su u opisnoj datoteci.
  - Treba pročitati datoteku te ažurirati artikle.



# Zadaci za vježbu

- ❑ Napisati aplikaciju za prijenos podataka iz *Excel* datoteke u bazu podataka *Firma*. Izvorni podaci su:

A	B	C	D	E	F
ImeOsobe	PrezimeOsobe	JMBG	AdrPartnera	NazMjestaPartnera	PostBrMjestaPartnera
Biljana	Ljubešić	1001971330031	Borovik 10	Zagreb	10000
Vladimir	Linke	0603941330092	Tržnica Utrine	Zagreb	10000

G	H	I
AdriIsporuke	NazMjestaliIsporuke	PostBrMjestaliIsporuke
Borovik 10	Zagreb	10000
Dalmatinska 2	Zagreb	10000

- ❑ *AdrPartnera*, *AdriIsporuke*, te pripadni *IdMjestaPartnera* i *IdMjestaliIsporuke* (pročitani iz tablice *Mjesto* na temelju *NazMjesta* i *PostBrMjesta*) treba kopirati u tablicu *Partner*.
- ❑ *IdPartnera* je *identity*, a *TipPartnera* 'O'.
- ❑ *ImeOsobe*, *PrezimeOsobe* i *JMBG* trebaju biti prebačeni u tablicu *Osoba* (*IdOsobe* treba biti jednak *IdPartner*)

# XML

---

# XML

## ❑ XML - eXtensible Markup Language

- Jezik koji omogućuje logičko strukturiranje podataka
- Općenita specifikacija jezika za opisivanje podataka
  - sintaksa i gramatika za izradu Document Type Definitions (DTD)
  - standard definiran od strane World Wide Web Consortium (W3C)
  - <http://www.w3.org/TR/2000/REC-xml-20001006>
- Struktura i podaci pohranjeni u obliku teksta

## ❑ Metapodaci

- podaci koji opisuju druge podatke, podaci sa semantikom
- za definiranje podatkovnih struktura koriste se oznake (*tags*) slično HTML
  - XML izgleda slično HTML ali nije HTML !
- opisuju se samo podaci ali ne i način njihovog prikaza

## ❑ Primjer: ADO \ DokumentStavka – DataSetDokumentStavka.xsd

- pogled u dizajneru i pogled u uređivaču teksta



# XML oznake

## ☐ Standardne oznake

- početak `<oznaka>`
- završetak `</oznaka>`  
ili `/>`
- prazni element `<oznaka/>`

## ☐ Elementi

- mogu gnijezditi druge elemente, ali smije postojati samo jedan korijen
- mogu imati pridružene attribute
- mogu, a ne moraju, imati vrijednosti

## ☐ Atributi

- svojstva elemenata

## ☐ XML razlikuje velika i mala slova u nazivima elemenata!

## ☐ Posebne oznake

- Deklaracija dokumenta: `<?xml ... ?>`
- Komentar: `<!-- opaska -->`
- Document type declaration (DTD):
  - `<! DOCTYPE [ ... ]>`
- Definition of document elements in an Internal DTD:
  - `<!ELEMENT >`, `<!ATTLIST>`, etc.

```
<?xml version="1.0" encoding="utf-8" ?>
<korijen>
  <element
    atribut="vrijednost" ...
    <atribut>vrijednost</atribut>
  </element>
</korijen>
```

## ☐ Kako nastane XML (datoteka) ?

# Primjena XML

- Definiranje .NET skupova podataka (*DataSet*)
- Rezultati web servisa, npr. <http://www.webservice.net/globalweather.aspx>
- Prijenos podataka, npr. <http://www.pbz.hr/ExchangeRateList/?lang=hr>
- Dokumentiranje programa
- ...

## ☐ SQL naredba koja vraća rezultat u XML formatu

- SELECT ... FOR XML mode [, XMLDATA] [, ELEMENTS][, BINARY BASE64]
- mode:
  - RAW (generički redak),
  - AUTO (redak tablice),
  - EXPLICIT (specifikacija za ugniježdene zavisne podatke)
- XMLDATA – rezultat sadrži shemu podataka i podatke
- ELEMENTS – stupci se tretiraju kao podelementi, a ne kao atributi XML čvorova, može se koristiti samo uz AUTO
- BINARY BASE64 – binarni podaci su BASE64 kodirani

## ☐ FOR XML se ne može koristiti u podupitima !

# Jednostavni SQL upiti s FOR XML modifikatorom

❑ **SELECT \* FROM Mjesto FOR XML AUTO**

```
<mjesto IdMjesta="132" OznDrzave="HR" NazMjesta="Bast"  
PostBrMjesta="21320" PostNazMjesta="Baška Voda" />  
<mjesto IdMjesta="133" OznDrzave="HR" NazMjesta="Bastajski  
Brđani" PostBrMjesta="43531" PostNazMjesta="Veliki Bastaji" />  
...
```

❑ **SELECT \* FROM Mjesto FOR XML RAW**

```
<row IdMjesta="132" OznDrzave="HR" NazMjesta="Bast"  
PostBrMjesta="21320" PostNazMjesta="Baška Voda" />  
<row IdMjesta="133" OznDrzave="HR" NazMjesta="Bastajski Brđani"  
PostBrMjesta="43531" PostNazMjesta="Veliki Bastaji" />  
...
```

# Ugniježdeni elementi u rezultatu spojnog upita

```
❑ SELECT * FROM Drzava INNER JOIN Mjesto  
ON Drzava.OznDrzave=Mjesto.OznDrzave FOR XML AUTO
```

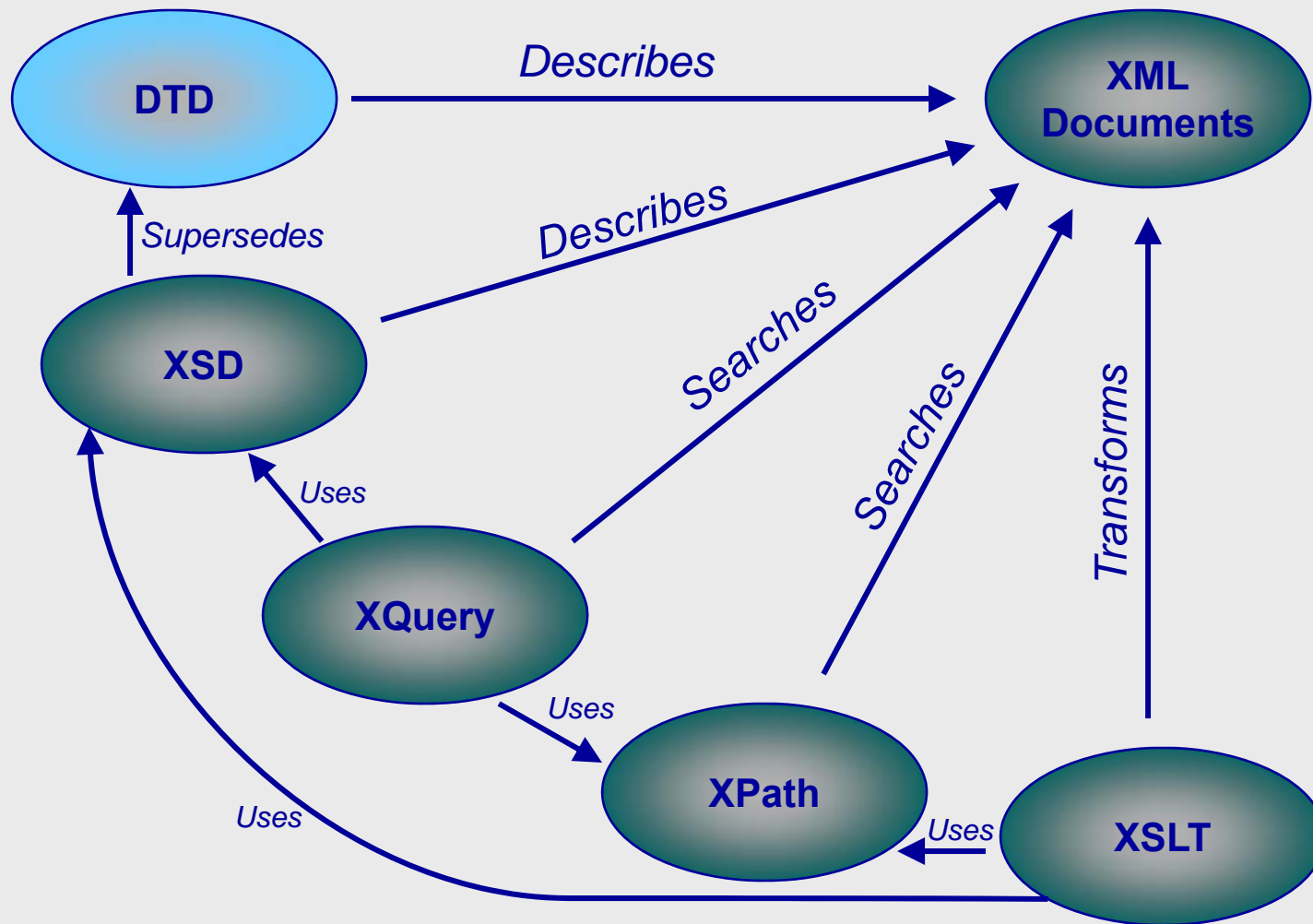
```
<drzava OznDrzave="HR" NazDrzave="Croatia" ISO3Drzave="HRV"  
SifDrzave="191">  
  <mjesto IdMjesta="132" OznDrzave="HR" NazMjesta="Bast"  
PostBrMjesta="21320" PostNazMjesta="Baška Voda" />  
  <mjesto IdMjesta="133" OznDrzave="HR" NazMjesta="Bastajski  
Brđani" PostBrMjesta="43531" PostNazMjesta="Veliki Bastaji" />  
  <mjesto IdMjesta="134" OznDrzave="HR" NazMjesta="Bašana"  
PostBrMjesta="23250" PostNazMjesta="Pag" />  
  ...  
</drzava>
```

# Primjer rezultata sa shemom podataka i podacima

❑ **SELECT \* FROM Mjesto FOR XML AUTO, XMLDATA, ELEMENTS**

```
<Schema name="Schema2" xmlns="urn:schemas-microsoft-com:xml-  
data" xmlns:dt="urn:schemas-microsoft-com:datatypes">  
  <ElementType name="mjesto" content="textOnly" model="closed"  
order="many">  
    <element type="IdMjesta" />  
    <element type="OznDrzave" />  
    ...  
  </ElementType>  
  <ElementType name="IdMjesta" ... dt:type="i4" />  
  <ElementType name="OznDrzave" ... dt:type="string" />  
</Schema>  
<mjesto xmlns="x-schema:#Schema2">  
  <IdMjesta>132</IdMjesta>  
  <OznDrzave>HR</OznDrzave>  
  <NazMjesta>Bast</NazMjesta>  
</mjesto>  
...
```

# Vrste XML dokumenata



# Obrada XML dokumenata

## ❑ XSD - XML schema

- jezik za opisivanje podataka s vlastitom sintaksom i gramatikom
- za opisivanje podataka koristi se XML → alternativa DTD

## ❑ XPath – jezik za adresiranje dijelova XML dokumenta

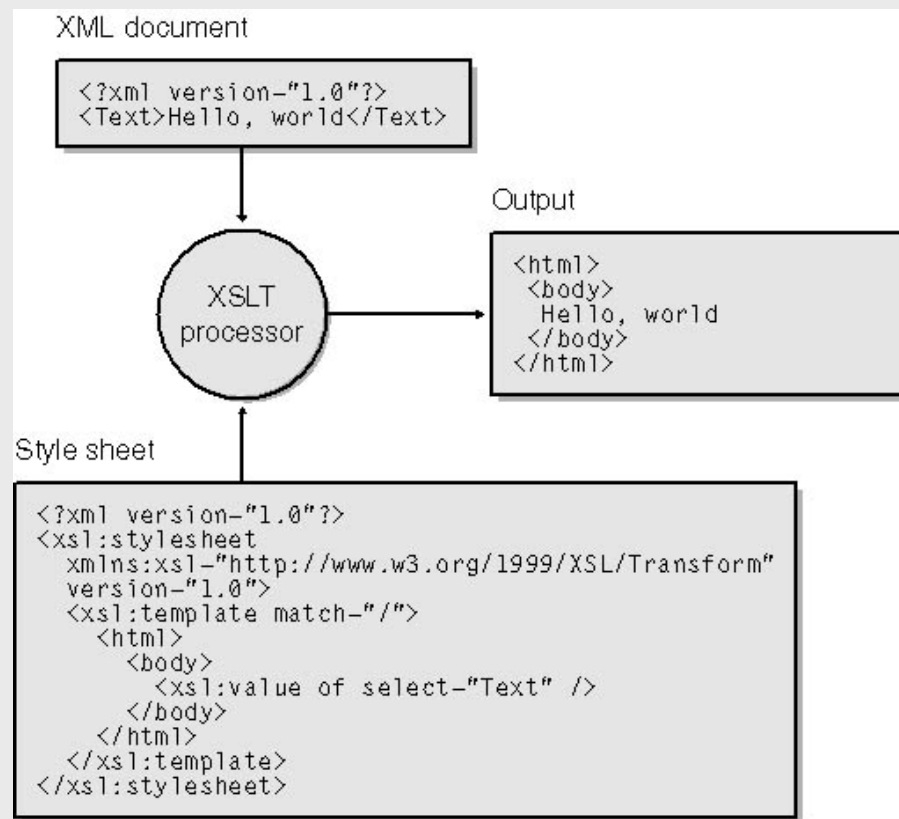
- omogućuje selekciju čvorova u XML dokumentu
- koristi se notacija slična onoj za definiranje URL

## ❑ XSL - Extensible Stylesheet Language

- jezik za definiranje popisa stilova

## ❑ XSLT - XSL Transformations

- jezik za transformaciju XML dokumenata
- XML programski jezik – ima pravila, izračunava izraze itd.



# Jednostavni tipovi podataka

## ❑ Jednostavni : <simpleType>

- deklaracija

- <simpleType name = "FirstName" type = "string"/>

- ne mogu sadržavati pod-elemente ili attribute

- mogu deklarirati ograničenja

- minLength, maxLength, length, etc

- mogu se koristiti kao osnovni tipovi za složene tipove

## ❑ Primjer

```
<simpleType name = "Ime">
  <restriction base = "string">
    <minLength value = "0" />
    <maxLength value = "25" />
  </restriction>
</simpleType>
```



# Složeni tipovi podataka

## ❑ Složeni: <complexType>

- zasnivaju se na jednostavnim tipovima ili na postojećim složenim tipovima
- mogu deklarirati elemente ili reference elemenata
- mogu deklarirati attribute ili reference na grupe podataka

```
<complexType name= "Partner">
  <sequence>
    <element name= "Naziv" type="Name" />
    <element name= "Adresa" type="Address" />
  </sequence>
</complexType>

<complexType name="Address">
  <sequence>
    <element name="Ulica" type="string" />
    <element name="Mjesto" type="string" />
    <element name="Posta" type="string" />
  </sequence>
</complexType>
```

# Kreiranje XML sheme u Visual Studio .NET

## ❑ Kreiranje XML sheme u Visual Studio .NET

- dodavanje sheme: Project / Add New Item / XML Schema
- uređivanje sheme: desni klik, Open ili View
- dodavanje čvorova (Element, Key, Relation, ..)
  - Server Explorer / Data Connections / Connection / Tables / drag&drop
  - ToolBox / drag&drop
  - Schema Designer / Add ...

## ❑ Primjer: Datoteke \ TecajnaLista – XMLSchema1

# Prostor imena *System.Xml*

## System.Xml

.Xsl

EntityHandling  
Formatting  
NameTable  
ReadState  
TreePosition  
Validation  
WriteState  
XmlAttribute  
XmlAttributeCollection  
XmlCDataSection  
XmlCharacterData

.XPath

XmlCharType  
XmlComment  
XmlConvert  
XmlDataDocument  
XmlDeclaration  
XmlDocument  
XmlDocumentFragment  
XmlDocumentType  
XmlElement  
XmlEntity  
XmlEntityReference  
XmlNamedNodeMap

.Serialization

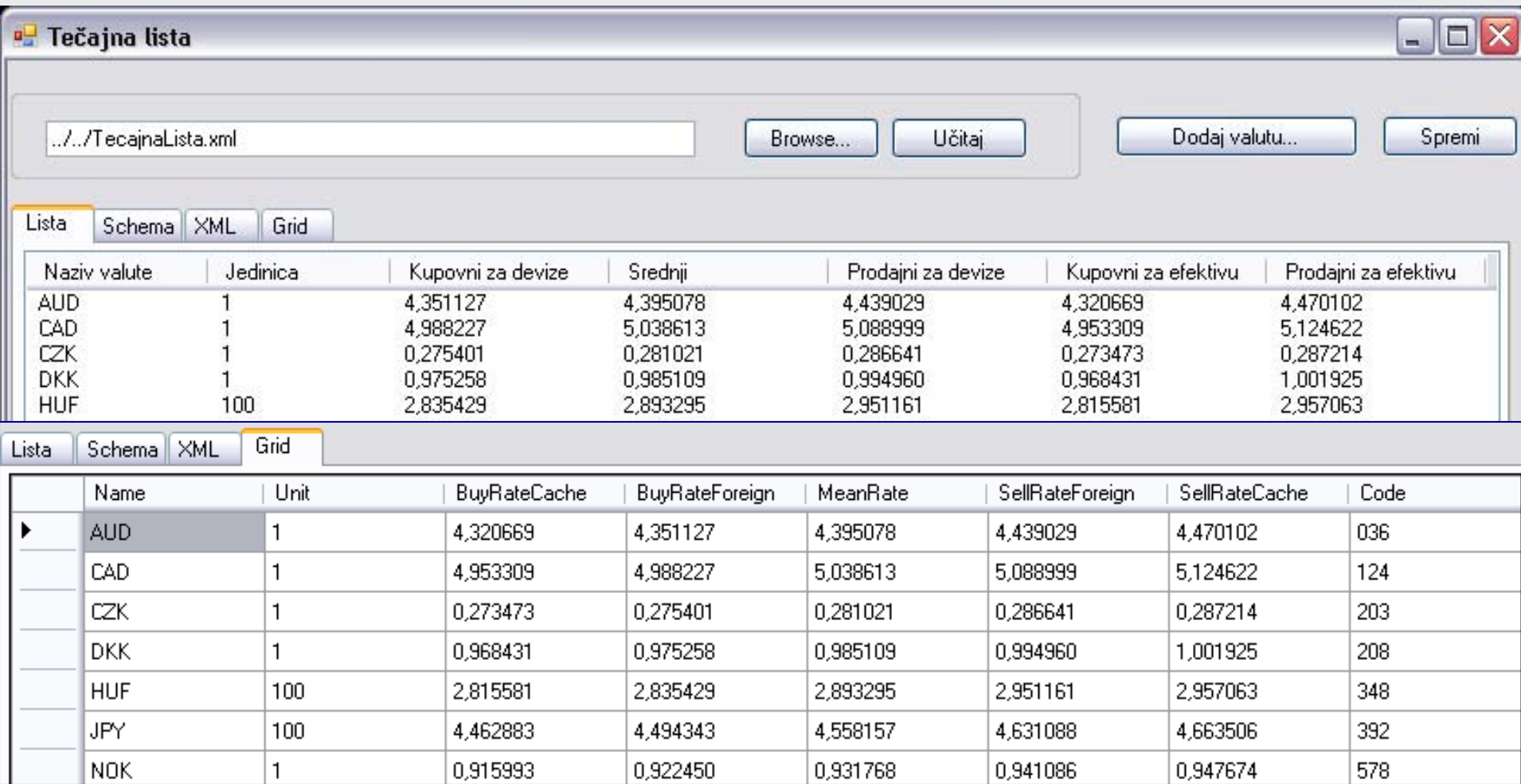
.Schema

XmlNode  
XmlNodeReader  
XmlNodeType  
XmlNotation  
XmlReader  
XmlSpace  
XmlText  
XmlTextReader  
XmlTextWriter  
XmlUriResolver  
XmlWhitespace  
XmlWriter  
...

# Primjer: Tečajna lista

## ❑ Primjer: Datoteke \ TecajnaLista

- ulazna datoteka TecajnaLista.xml
- podaci preuzeti s <http://www.pbz.hr/ExchangeRateList/?lang=hr>



Naziv valute	Jedinica	Kupovni za devize	Srednji	Prodajni za devize	Kupovni za efektivu	Prodajni za efektivu
AUD	1	4,351127	4,395078	4,439029	4,320669	4,470102
CAD	1	4,988227	5,038613	5,088999	4,953309	5,124622
CZK	1	0,275401	0,281021	0,286641	0,273473	0,287214
DKK	1	0,975258	0,985109	0,994960	0,968431	1,001925
HUF	100	2,835429	2,893295	2,951161	2,815581	2,957063

Name	Unit	BuyRateCache	BuyRateForeign	MeanRate	SellRateForeign	SellRateCache	Code
AUD	1	4,320669	4,351127	4,395078	4,439029	4,470102	036
CAD	1	4,953309	4,988227	5,038613	5,088999	5,124622	124
CZK	1	0,273473	0,275401	0,281021	0,286641	0,287214	203
DKK	1	0,968431	0,975258	0,985109	0,994960	1,001925	208
HUF	100	2,815581	2,835429	2,893295	2,951161	2,957063	348
JPY	100	4,462883	4,494343	4,558157	4,631088	4,663506	392
NOK	1	0,915993	0,922450	0,931768	0,941086	0,947674	578

# Razredi *XMLDocument* i *XMLNode*

## ❑ Neki postupci *XMLDocument*

- `AppendChild`, `RemoveChild`, `ReplaceChild` – rukovanje čvorovima
- `CreateAttribute` – kreira atribut
- `CreateElement` – kreira element
- `CreateComment` – kreira komentar
- `CreateNode` – kreira instancu razreda `XMLNode`
- `Load` – učitava XML tekst iz navedenog *Stream*, *TextReader* ili *XMLReader*
- `Save` – sprema sadržaj na navedenu lokaciju
- `GetElementsByTagName` – vraća listu čvorova po zadanoj oznaci
- `SelectNodes`, `SelectSingleNode` – traženje čvorova XPath izrazom

## ❑ Neki postupci *XMLNode*

- `AppendChild`, `InsertAfter`, `InsertBefore` – rukovanje čvorovima

## ❑ Neka svojstva *XMLNode*

- `FirstChild`, `LastChild` – prvo i zadnje dijete
- `NextSibling`, `PreviousSibling` – slijedeći i prethodni čvor
- `InnerText` – vrijednost čvora i njegove djece

# Primjer čitanja XML dokumenta

## ❏ Primjer: Datoteke \ TecajnaLista

```
listView1.Items.Clear();

XmlDocument tecajnaLista = new XmlDocument();
tecajnaLista.Load(imeDatoteke);
XmlNodeList listaValuta =
    tecajnaLista.GetElementsByTagName("Currency");

foreach (XmlNode cvor in listaValuta)
{
    XmlElement valuta = (XmlElement)cvor;

    XmlNode nazivValute = valuta.GetElementsByTagName("Name")[0];
    ...
    ListViewItem item = new ListViewItem(nazivValute.InnerText);
    ...
    listView1.Items.Add(item);
}
```

# Primjer čitanja XML dokumenta - alternativa

## ❏ Primjer: Datoteke \ TecajnaLista

```
listView1.Items.Clear();

XmlDocument tecajnaLista = new XmlDocument();
tecajnaLista.Load(imeDatoteke);
XmlNodeList listaValuta =
    tecajnaLista.GetElementsByTagName("Currency");

foreach (XmlNode cvor in listaValuta)
{
    item = null;
    foreach (XmlNode el in cvor.ChildNodes)
    {
        if (item == null)
            item = new ListViewItem(el.InnerText);
        else
            item.SubItems.Add(el.InnerText);
    }
    listView1.Items.Add(item);
}
```

# Kreiranje XML elementa

```
private XmlElement NewElement(  
    XmlDocument xmlDokument, string nazivElementa, string text)  
{  
    XmlElement xmlElement = xmlDokument.CreateElement(nazivElementa) ;  
    XmlText xmlText = xmlDokument.CreateTextNode(text) ;  
    xmlElement.AppendChild(xmlText) ;  
    return xmlElement ;  
}
```

```
XmlElement nazivValute  
= NewElement(tecajnaLista,  
    "Name", textBoxNaziv.Text);
```

**Dodavanje nove valute**

Naziv valute:	HRK
Jedinica:	1
Kupovni za efektivu:	1
Kupovni za devize:	1
Srednji:	1
Prodajni za devize:	1
Prodajni za efektivu:	1

**Dodaj**



# Rukovanje strukturom XML dokumenta

## ❏ Primjer: Datoteke \ TecajnaLista – FormDodajValutu

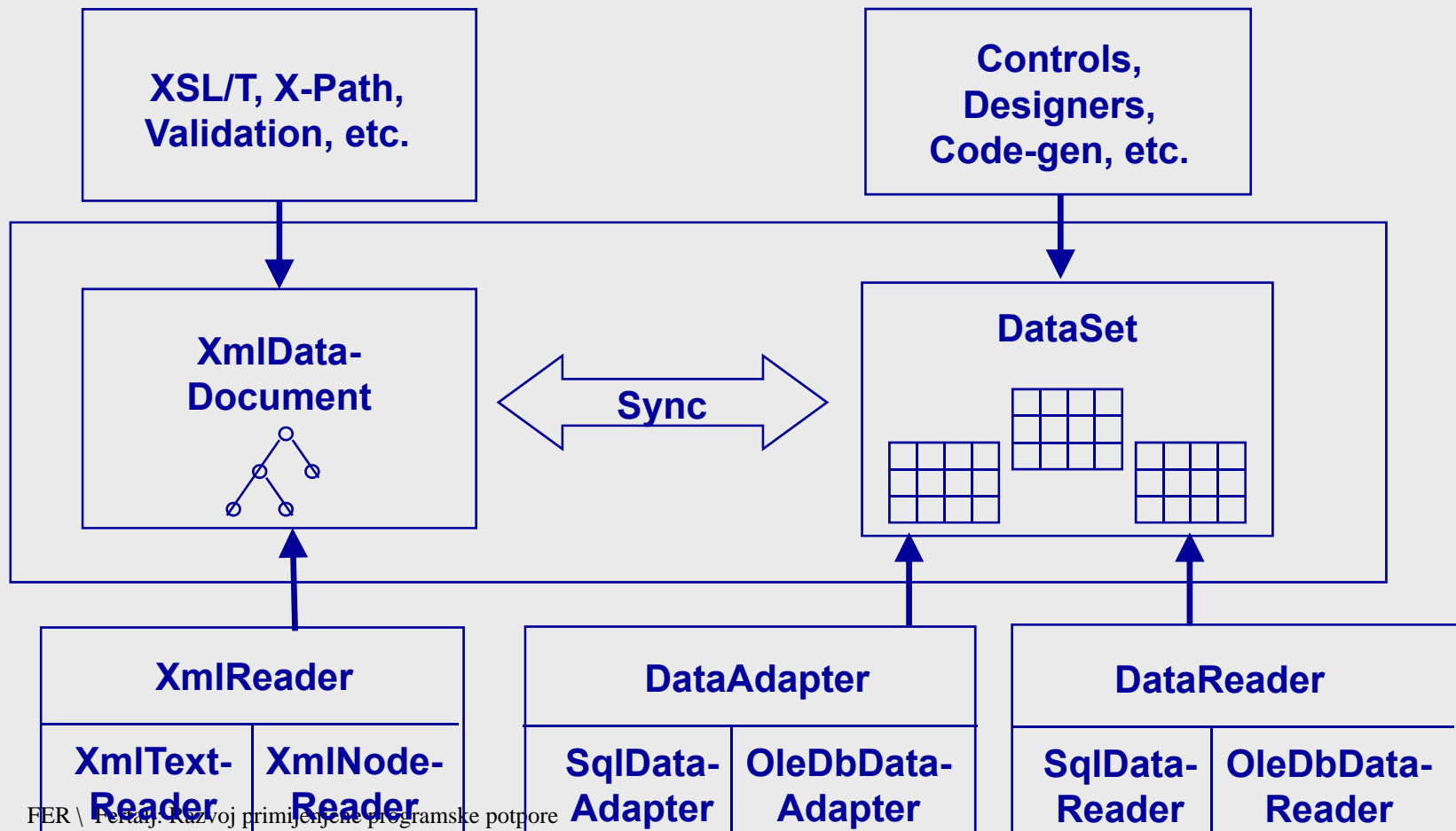
```
private void buttonDodaj_Click(object sender, EventArgs e)
{
    XmlDocument tecajnaLista = new XmlDocument();
    tecajnaLista.Load(imeDatoteke);

    XmlElement novaValuta = tecajnaLista.CreateElement("Currency");

    XmlElement nazivValute
        = NewElement(tecajnaLista, "Name", textBoxNaziv.Text);
    XmlElement jedinica
        = NewElement(tecajnaLista, "Unit", textBoxJedinica.Text);
    ...
    novaValuta.AppendChild(nazivValute);
    novaValuta.AppendChild(jedinica);
    ...
    tecajnaLista.DocumentElement.FirstChild.AppendChild(novaValuta);
    tecajnaLista.Save(imeDatoteke);
}
```

# XML i DataSet

- ❑ Razred *XmlDataDocument* – omogućuje čitav XML dokument u memoriji računala koji se može koristiti kao DataSet
  - Izveden iz razreda *XmlNode*,
  - Svojstvo `DataSet` – relacijska reprezentacija podataka u dokumentu



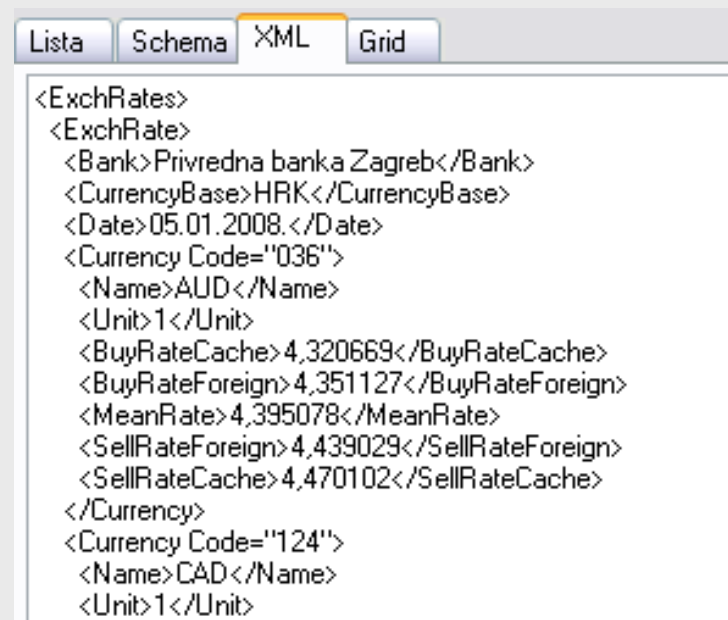
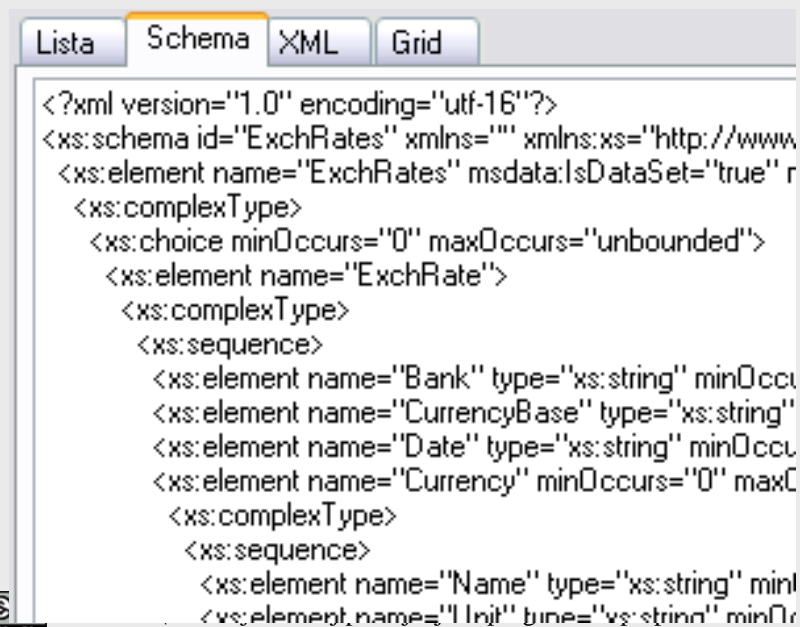
# ***DataSet*** postupci za rad s XML-om

- ❑ **ReadXMLSchema** – učitava definiciju iz XSD sheme ili iz XML-a
  - `ReadXMLSchema(stream) // string, TextReader, XmlReader`
- ❑ **ReadXML** – učitava XML podatke u DataSet
  - `ReadXML(stream, XmlReadMode)`
    - `XmlReadMode` – `Auto`, `ReadSchema`, `IgnoreSchema`, `InferSchema`, `DiffGram`, `Fragment`
- ❑ **WriteXMLSchema** – zapisuje XML shemu
  - `WriteXMLSchema(stream) // string, TextReader, XmlReader...`
- ❑ **WriteXML** – zapisuje podatke
  - `WriteXML(stream, XmlWriteMode)`
  - `XmlWriteMode` – `IgnoreSchema`, `WriteSchema`, `DiffGram`
    - `IgnoreSchema=False` za veze s postavljenim atributom `Nested=true` podatke zapisuje hijerarhijski
- ❑ **InferXMLSchema** – izlučuje shemu iz strukture XML podataka
  - `InferXMLSchema(stream) // file, TextReader ili XmlReader`
- ❑ **GetXMLSchema, GetXML** – string sheme odnosno podataka

# Učitavanje XML sheme i podataka

## ❑ Primjer: Datoteke \ TecajnaLista

```
XmlDataDocument xmlLista = new XmlDataDocument();  
xmlLista.DataSet.ReadXml(imeDatoteke, XmlReadMode.InferSchema);  
  
textBoxXMLSchema.Text = xmlLista.DataSet.GetXmlSchema();  
textBoxXML.Text = xmlLista.DataSet.GetXml();  
  
dataGridView1.DataSource = xmlLista.DataSet;  
dataGridView1.DataMember = xmlLista.DataSet.Tables[1].TableName;
```



# Zapisivanje XML sheme i podataka

## ❏ Primjer: Datoteke \ TecajnaLista

```
private void buttonSpremi_Click(object sender, EventArgs e)
{
    ((DataSet)dataGridView1.DataSource).WriteXml(
        @"c:\projects\tecajna_xsd.xml", XmlWriteMode.WriteSchema);
    ((DataSet)dataGridView1.DataSource).WriteXml(
        @"c:\projects\tecajna.xml", XmlWriteMode.IgnoreSchema);
    MessageBox.Show(@"c:\projects\tecajna.* Ok");
}
```

# Razredi *XmlReader* i *XmlWriter*

## ❑ *XmlReader*

- apstraktni razred za čitanje XML,
- forward-only, non-cached XML stream reader
- svojstva
  - Value: vrijednost čvora
  - NodeType: tip čvora
  - HasValue: oznaka da čvor ima vrijednost
  - LocalName: naziv čvora bez prefiksa
  - ReadState: Closed, EndOfFile, Error, ...

## ❑ *XMLWriter*

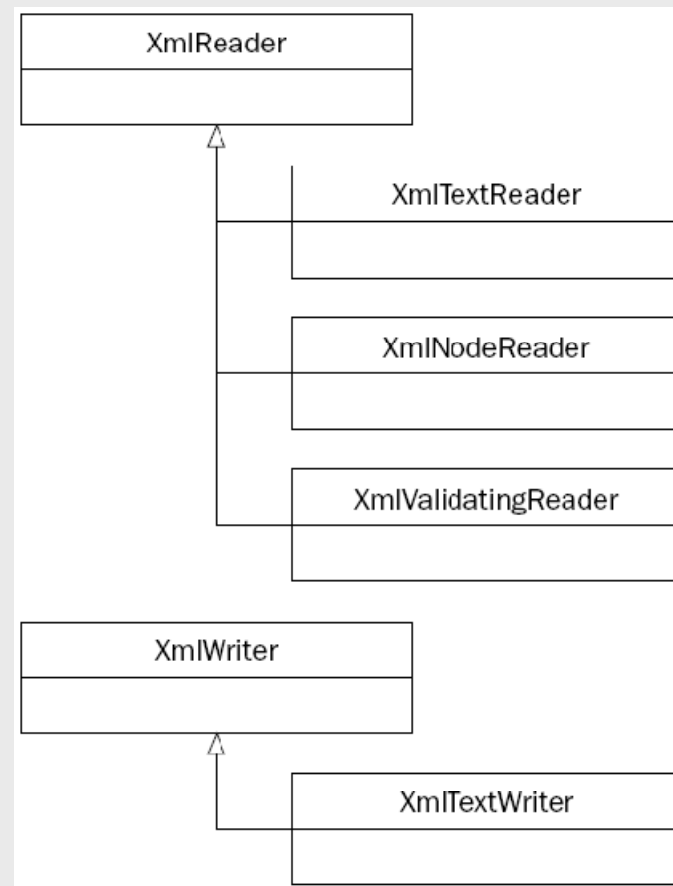
- apstraktni razred za pisanje XML
- forward-only, non-cached XML stream writer
- svojstva
  - WriteState: Attribute, Content, Element, ...
  - XmlLang: vraća aktualni xml:lang
  - XmlSpace: vraća aktualni xml:space

## ❑ *XmlTextReader* – izveden iz *XMLReader*

- postupci: Read, MoveToElement, ReadString

## ❑ *XmlTextWriter* – izveden iz *XMLWriter*

- postupci: WriteDocType, WriteComment, WriteName



# Primjer zasebnog učitavanja sheme i podataka

## ❑ Primjer za *XMLReader* i *XMLDataDocument.Load*

```
// Create an XmlDocument.  
XmlDataDocument doc = new XmlDocument();  
  
// Load the schema file.  
doc.DataSet.ReadXmlSchema("tecajna.xsd");  
  
// Load the XML data.  
XmlTextReader reader = new XmlTextReader("tecajna.xml");  
reader.MoveToContent(); // Moves the reader to the root node.  
doc.Load(reader);
```

# Serijalizacija

---



# Serijalizacija

## ❑ Serialization – proces zapisivanja stanja objekata na medij pohrane


## ❑ Binarna serijalizacija

- naziv klase, javna i privatna polja objekta te assembly koji sadrži klasu budu konvertirani u tok bajtova (stream)
- Razred ***BinaryFormatter*** – postupci:
  - `public void Serialize(Stream serializationStream, object graph);`
  - `public object Deserialize(Stream serializationStream);`

## ❑ XML serijalizacija

- samo javna polja i svojstva budu serijalizirani u XML formatu
- Razred ***XmlSerializer*** – postupci:
  - `public void Serialize(TextWriter textWriter, object o);`
  - `public object Deserialize(Reader textReader);`

# Primjer razreda koji se serijaliziraju

- ❑ **Primjer:**  **Datoteke \ Serialize – razredi**
  - serijaliziramo kolekciju trokuta definiranih s po 3 točke
- ❑ **Atribut *Serializable*** – određuje da klasa može biti serijalizirana
- ❑ **Sučelje *ISerializable*** – omogućuje objektu kontrolirati serijalizaciju

```
[Serializable()] // točka definirana s 2 koordinate  
public class XYPoint  
    : System.Runtime.Serialization.ISerializable
```

```
[Serializable()] // trokut definiran s 3 točke  
public class Triangle
```

```
[Serializable()] // kolekcija trokuta  
public class TriangleCollection  
    : System.Collections.CollectionBase
```

# Primjer binarne serijalizacije

## ❏ Primjer: 📁 Datoteke \ Serialize - Form

```
private void saveBinary_Click(object sender, System.EventArgs e)
{
    System.IO.Stream stream = new System.IO.FileStream(
        binaryFile, System.IO.FileMode.Create);
    BinaryFormatter binary = new BinaryFormatter();
    binary.Serialize(stream, triangles);
    stream.Close();
    ...
}

private void loadBinary_Click(object sender, System.EventArgs e)
{
    System.IO.Stream stream = new System.IO.FileStream(
        binaryFile, System.IO.FileMode.Open);
    BinaryFormatter binary = new BinaryFormatter();
    triangles = (TriangleCollection)binary.Deserialize(stream);
    stream.Close();
    ...
}
```

# Primjer XML serijalizacije

## ❏ Primjer: Datoteke \ Serialize - Form

```
private void saveXML_Click(object sender, System.EventArgs e)
{
    System.IO.TextWriter writer =
        new System.IO.StreamWriter(xmlFile);
    XmlSerializer xmlSerial =
        new XmlSerializer(typeof(TriangleCollection));
    xmlSerial.Serialize(writer, triangles);
    writer.Close();
    ...
private void loadXML_Click(object sender, System.EventArgs e)
{
    System.IO.TextReader reader =
        new System.IO.StreamReader(xmlFile);
    XmlSerializer xmlSerial =
        new XmlSerializer(typeof(TriangleCollection));
    triangles =
        (TriangleCollection)xmlSerial.Deserialize(reader);
    reader.Close();
}
```

# Postupci objekta korišteni pri (de)serijalizaciji

## ❏ Primjer: Datoteke \ Serialize - XYpoint

```
public void GetObjectData( // poziva se pri serijalizaciji
    System.Runtime.Serialization.SerializationInfo info,
    System.Runtime.Serialization.StreamingContext context)
{
    info.AddValue("X", x);
    info.AddValue("Y", y);
}

public XYPoint( // konstruktor pri deserijalizaciji
    System.Runtime.Serialization.SerializationInfo info,
    System.Runtime.Serialization.StreamingContext context)
{
    x = info.GetInt32("X");
    y = info.GetInt32("Y");
}
```

# Reference

## ☐ C# School Book – Free ebook

- <http://www.programmersheaven.com/2/CSharpBook>

☐ <http://www.w3.org/XML/>

☐ <http://www.w3schools.com/xml/>

☐ <http://msdn.microsoft.com/xml/>

☐ <http://www.xml.com/>

☐ <http://www.xmlfiles.com/xml/>