# LINQ-To-XML

# W3C DOM & LINQ-To-XML

- W3C Document Object Model (DOM)
  - W3C: World Wide Web Consortium är en internationell organisation som utvecklar öppna standarder för Internet
  - Traditionellt API för XML-filer
  - ”Det objektorienterade sättet”
- LINQ-To-XML
  - .NET specifikt
  - Moderniserat och lättare API
  - ”Det funktionella sättet”

# Objektorienterat
# Skapa XML-träd nedifrån och upp

```
XmlDocument XmlDoc = new XmlDocument();

XmlElement title = XmlDoc.CreateElement("Title");
title.InnerText = "Pulp Fiction";
XmlElement year = XmlDoc.CreateElement("Year");
year.InnerText = "1994";
XmlElement rating = XmlDoc.CreateElement("Rating");
rating.InnerText = "9.0";
XmlElement userVotes = XmlDoc.CreateElement("UserVotes");
userVotes.InnerText = "546812";

XmlElement movie = XmlDoc.CreateElement("Movie");
movie.AppendChild(title);
movie.AppendChild(year);
movie.AppendChild(rating);
movie.AppendChild(userVotes);

XmlElement crime = XmlDoc.CreateElement("Category");
crime.InnerText = "Crime";
XmlElement thriller = XmlDoc.CreateElement("Category");
thriller.InnerText = "Thriller";

XmlElement categories = XmlDoc.CreateElement("Categories");
categories.AppendChild(crime);
categories.AppendChild(thriller);
movie.AppendChild(categories);

XmlElement movies = XmlDoc.CreateElement("Movies");
movies.AppendChild(movie);
XmlDoc.AppendChild(movies);
```

```
<Movies>
  <Movie>
    <Title>Pulp Fiction</Title>
    <Year>1994</Year>
    <Rating>9.0</Rating>
    <UserVotes>546812</UserVotes>
    <Categories>
      <Category>Crime</Category>
      <Category>Thriller</Category>
    </Categories>
  </Movie>
</Movies>
```

# Functional construction

```
XElement xDoc =
    new XElement("Movies",
```

▲ 5 of 5 ▼ XElement.XElement(XName name, **params object[] content**)

Initializes a new instance of the System.Xml.Linq.XElement class with the specified name and content.

***content:*** *The initial content of the element.*

- ”content” parametern är flexibel och kan ta emot alla datatyper vilket möjliggör ”functional construction”, datatyperna tolkas olika exempelvis:
  - String : läggs till som värde till elementet
  - XElement : läggs till som barn-element
  - Xattribute : läggs till som element-attribut
  - IEnumerable : Reglerna appliceras rekursivt för alla element
  - Null : Ignoreras
  - För okända datatyper anropas ToString()-metoden
- Nyckelordet params tillåter oss att ange ett godtyckligt antal parametrar antigen som komma-separerade värden eller i form av en array av den specificerade datatypen.

# Functional construction

```
// Linq-2-XML: Functional construction
XElement xDoc =
    new XElement("Movies",
        new XElement("Movie",
            new XElement("Title", "Pulp Fiction"),
            new XElement("Year", 1994),
            new XElement("Rating", 9.0),
            new XElement("UserVotes", 546812),
            new XElement("Categories",
                new XElement("Category", "Crime"),
                new XElement("Category", "Thriller"))));
```

```
<Movies>
  <Movie>
    <Title>Pulp Fiction</Title>
    <Year>1994</Year>
    <Rating>9.0</Rating>
    <UserVotes>546812</UserVotes>
    <Categories>
      <Category>Crime</Category>
      <Category>Thriller</Category>
    </Categories>
  </Movie>
</Movies>
```

# Functional construction

```
// DOM: Document Object Model
XmlDocument XmlDoc = new XmlDocument();

XmlElement title = XmlDoc.CreateElement("T
title.InnerText = "Pulp Fiction";
XmlElement year = XmlDoc.CreateElement("Ye
year.InnerText = "1994";
XmlElement rating = XmlDoc.CreateElement("
rating.InnerText = "9.0";
XmlElement userVotes =
XmlDoc.CreateElement("UserVotes");
userVotes.InnerText = "546812";

XmlElement movie = XmlDoc.CreateElement("M
movie.AppendChild(title);
movie.AppendChild(year);
movie.AppendChild(rating);
movie.AppendChild(userVotes);

XmlElement crime = XmlDoc.CreateElement("Category");
crime.InnerText = "Crime";
XmlElement thriller = XmlDoc.CreateElement("Category");
thriller.InnerText = "Thriller";

XmlElement categories =
XmlDoc.CreateElement("Categories");
categories.AppendChild(crime);
categories.AppendChild(thriller);
movie.AppendChild(categories);

XmlElement movies = XmlDoc.CreateElement("Movies");
movies.AppendChild(movie);
XmlDoc.AppendChild(movies);
```

```
// Linq-2-XML: Functional construction
XElement xDoc =
    new XElement("Movies",
        new XElement("Movie",
            new XElement("Title", "Pulp Fiction"),
            new XElement("Year", 1994),
            new XElement("Rating", 9.0),
            new XElement("UserVotes", 546812),
            new XElement("Categories",
                new XElement("Category", "Crime"),
                new XElement("Category", "Thriller"))));
```

```
<Movies>
  <Movie>
    <Title>Pulp Fiction</Title>
    <Year>1994</Year>
    <Rating>9.0</Rating>
    <UserVotes>546812</UserVotes>
    <Categories>
      <Category>Crime</Category>
      <Category>Thriller</Category>
    </Categories>
  </Movie>
</Movies>
```

# Functional construction & Query Expressions

```
var movies = new[] {
    new {
        Title = "Pulp Fiction",
        Year = 1994,
        Rating = 9.0,
        UserVotes = 546812,
        Categories = new [] { "Crime", "Drama" }
    },
    new {
        Title = "Shrek",
        Year = 2001,
        Rating = 7.9,
        UserVotes = 213955,
        Categories = new [] { "Animation", "Adventure", "Comedy" }
    }
};
XElement doc =
    new XElement("Movies",
        from movie in movies
        select new XElement("Movie",
            new XElement("Title", movie.Title),
            new XElement("Year", movie.Year),
            new XElement("Rating", movie.Rating),
            new XElement("UserVotes", movie.UserVotes),
            new XElement("Categories",
                from category in movie.Categories
                select new XElement("Category", category))));
```

```
<Movies>
  <Movie>
    <Title>Pulp Fiction</Title>
    <Year>1994</Year>
    <Rating>9</Rating>
    <UserVotes>546812</UserVotes>
    <Categories>
      <Category>Crime</Category>
      <Category>Drama</Category>
    </Categories>
  </Movie>
  <Movie>
    <Title>Shrek</Title>
    <Year>2001</Year>
    <Rating>7.9</Rating>
    <UserVotes>213955</UserVotes>
    <Categories>
      <Category>Animation</Category>
      <Category>Adventure</Category>
      <Category>Comedy</Category>
    </Categories>
  </Movie>
</Movies>
```

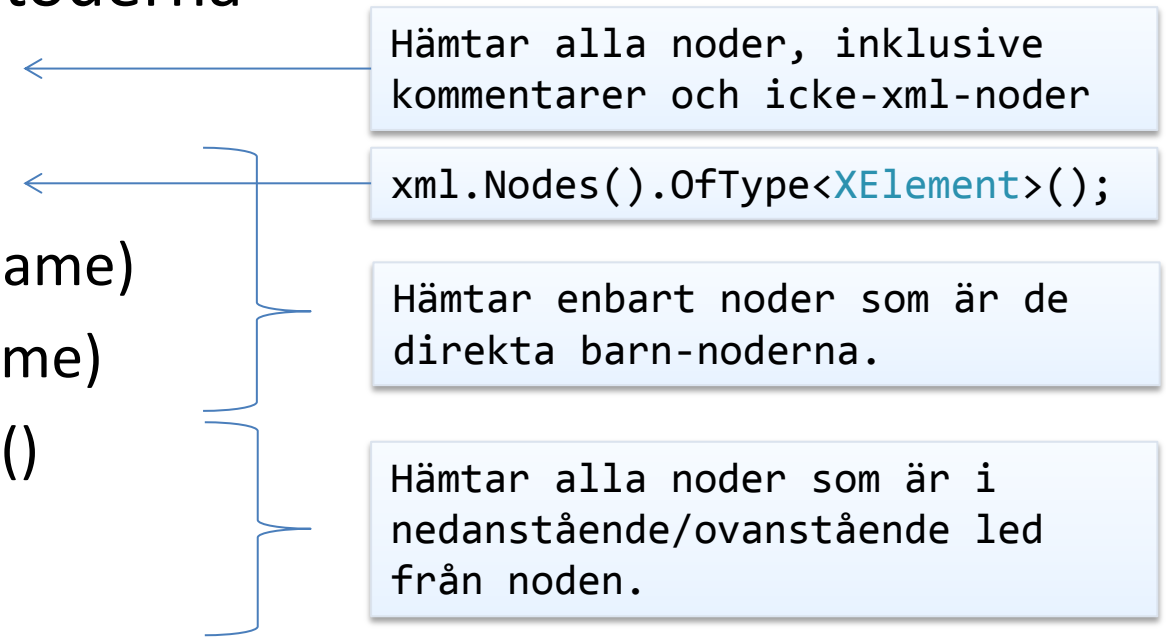
# Functional construction & Function Calls

```
XElement doc = new XElement("Movies", GetMovies(movies));

static IEnumerable<XElement> GetMovies(IEnumerable<Movie> movies)
{
    return
        from movie in movies
        select GetMovie(movie);
}

static XElement GetMovie(Movie movie)
{
    return
        new XElement("Movie",
            new XElement("Title", movie.Title),
            new XElement("Year", movie.Year),
            new XElement("Rating", movie.Rating),
            new XElement("UserVotes", movie.UserVotes),
            new XElement("Categories",
                GetCategoriesFromMovie(movie)));
}

static IEnumerable<XElement> GetCategoriesFromMovie(Movie movie)
{
    return
        from category in movie.Categories
        select new XElement("Category", category);
}
```

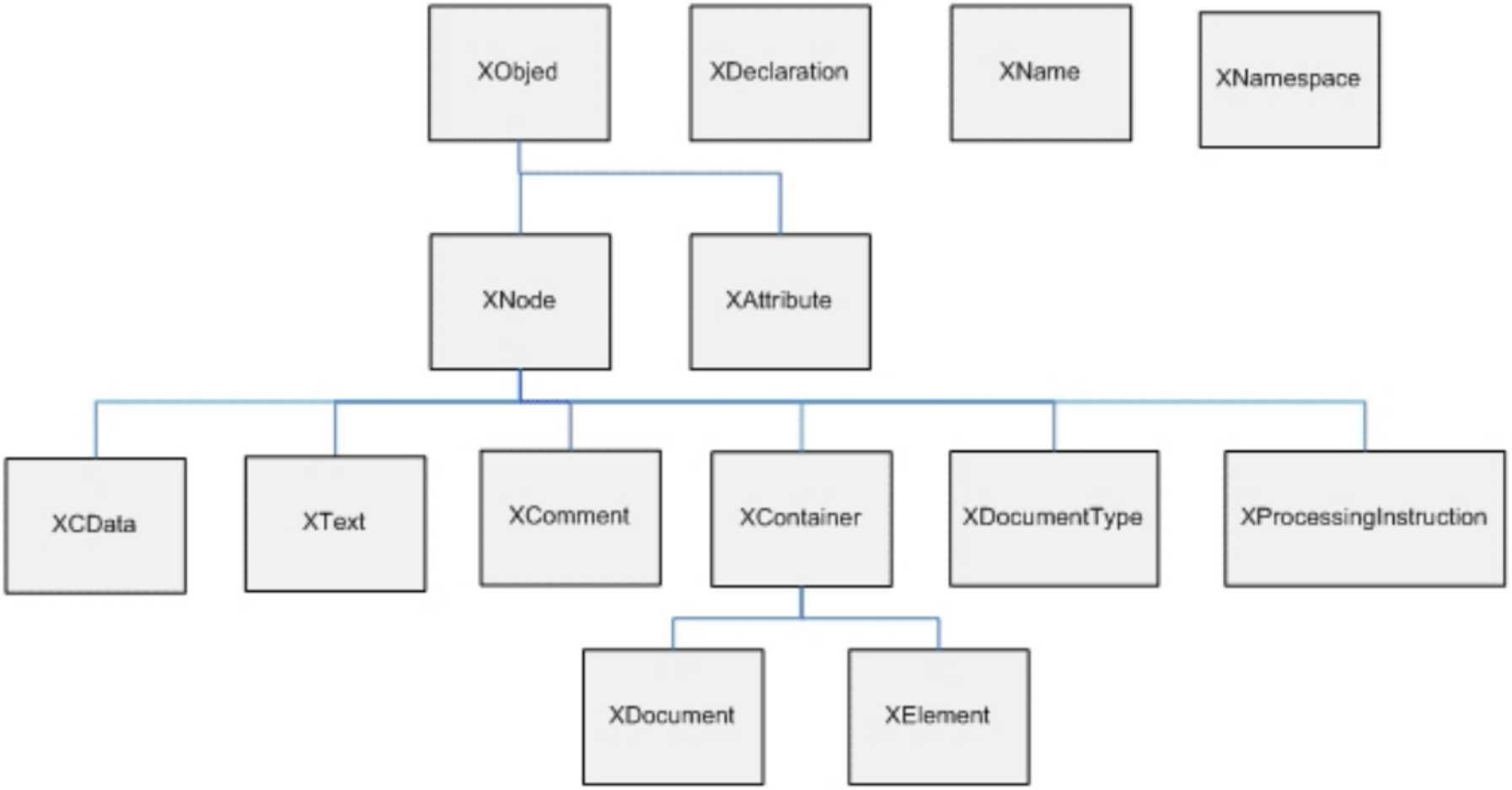
```
<Movies>
  <Movie>
    <Title>Pulp Fiction</Title>
    <Year>1994</Year>
    <Rating>9</Rating>
    <UserVotes>546812</UserVotes>
    <Categories>
      <Category>Crime</Category>
      <Category>Drama</Category>
    </Categories>
  </Movie>
  <Movie>
    <Title>Shrek</Title>
    <Year>2001</Year>
    <Rating>7.9</Rating>
    <UserVotes>213955</UserVotes>
    <Categories>
      <Category>Animation</Category>
      <Category>Adventure</Category>
      <Category>Comedy</Category>
    </Categories>
  </Movie>
</Movies>
```

# Traversera XML

- Navigera genom ett XML-träd med extension methods för IEnumerable<XElement>
- Vanligaste metoderna
  - Nodes()
  - Elements()
  - Elements(XName)
  - Element(Xname)
  - Descendants()
  - Ancestors()



# Klasshierarki
# LINQ-To-XML



# Traversera XML

```
var titles = movies.Elements("Movie").Elements("Title");
```

```
var titles = movies.Descendants("Title");
```

```
IEnumerable<XElement> titles:
[0]<Title>Pulp Fiction</Title>
[1]<Title>Shrek</Title>
```

```
var categories =
    movies.Elements("Movie").Elements("Categories");
```

```
IEnumerable<XElement> categories:
[0]
<Categories>
  <Category>Crime</Category>
  <Category>Drama</Category>
</Categories>
[1]
<Categories>
  <Category>Animation</Category>
  <Category>Adventure</Category>
  <Category>Comedy</Category>
</Categories>
```

```
<Movies>
  <Movie>
    <Title>Pulp Fiction</Title>
    <Year>1994</Year>
    <Rating>9</Rating>
    <UserVotes>546812</UserVotes>
    <Categories>
      <Category>Crime</Category>
      <Category>Drama</Category>
    </Categories>
  </Movie>
  <Movie>
    <Title>Shrek</Title>
    <Year>2001</Year>
    <Rating>7.9</Rating>
    <UserVotes>213955</UserVotes>
    <Categories>
      <Category>Animation</Category>
      <Category>Adventure</Category>
      <Category>Comedy</Category>
    </Categories>
  </Movie>
</Movies>
```

# Traversera XML
# Värden

- I löv-noderna finns det oftast värden av familjära datatyper som string, int, bool etc.
  - Ett naturligt sätt att utläsa värdet kan vara

```
double rating = double.Parse(
    movies.Element("Movie").Element("Rating").Value);
```

  - Det finns överlagrade explicita cast-operatorer för de vanligaste datatyperna, XElement -> double, int...

```
double rating =
    (double) movies.Element("Movie").Element("Rating");
```

```
string title =
    (string) movies.Element("Movie").Element("Title");
```

# XML Transformering

- Functional construction i samband med LINQ queries

```
XElement ratings =
    new XElement("Ratings",
        from rating in movies.Descendants("Rating")
        select rating);
```

```
XElement decentMovies =
    new XElement("DecentMovieTitles",
        from movie in movies.Elements("Movie")
        let rating = (double)movie.Element("Rating")
        where rating > 6.8
        select movie.Element("Title"));
```

# XML Transformering
# Nyckelordet let

- I query expressions har vi möjligheten att definiera variabler för temporära värden

```
from movie in movies.Elements("Movie")

let rating = (double)movie.Element("Rating")

where rating > 6.8
select movie.Element("Title"));
```

- Nyckelordet let eller en *let clause* är oföränderlig och kan ej förändras väl tilldelad ett värde

```
from number in oddNumbers
let temp = number
let temp2 = temp++
select numbe
```

Range variable 'temp' cannot be assigned to -- it is read only

# XML Transformering
# Nyckelordet let

- Strategier vid transformering av xml
- Minska upprepad kod / Förenkla kod
- Referera uppåt (data i förälder/farföräldrar) i xml strukturen
- Övergripande beräkningar som behövs ’för varje’

# Modifiera XML

```
XElement movie = new XElement("Movie");
XElement year = new XElement("Year", 2004);

movie.Add(year);
movie.Add(new XElement("Rating", 7.9),
          new XElement("UserVotes", 213955));

movie.AddFirst(new XElement("Title", "Shrek"));

movie.Element("UserVotes").AddBeforeSelf(
    new XComment("Wow, so many votes!"));

movie.Element("Year").ReplaceNodes(2001);

movie.Add(new XElement("Categories"));

movie.Element("Categories").ReplaceNodes(
    new XElement("Category", "Animation"),
    new XElement("Category", "Adventure"),
    new XElement("Category", "Comedy"));

movie.Nodes().OfType<XComment>().Remove();

movie.SetElementValue("Rating", 10.0);
movie.SetElementValue("Rating", null);
```

```
<Movie>
  <Title>Shrek</Title>
  <Year>2001</Year>
  <Rating>7.9</Rating>
  <UserVotes>213955</UserVotes>
  <Categories>
    <Category>Animation</Category>
    <Category>Adventure</Category>
    <Category>Comedy</Category>
  </Categories>
</Movie>
</Movie>
```

# XML-attribut

- XML-attribut representeras av klassen XAttribute
  - Ej noder utan oordnade nyckel-värde par som tillhör en nod
- Väldigt likt sätt att arbeta som med XElement

```
XElement movie =
    new XElement("Movie", new XAttribute("id", 757),
        new XElement("Title", "Shrek"));

// Get attribute
XAttribute id = movie.Attribute("id");

// Set attribute value
movie.SetAttributeValue("id", 12);

// Remove attribute
movie.Element("id").Remove();
```

# Läsa in XML

- Många olika sätt stödjs
  - Functional construction: Skapa XML i kod
  - Load: Från fil
  - Parse: Från en textsträng
  - Readers: XmlReader, TextReader

```
XElement movies = XDocument.Load(@"C:/movies.xml");

var movie = XElement.Parse(
    @"<Movie>
        <Title>Pulp Fiction</Title>
        <Year>1994</Year>
        <Rating>9</Rating>
        <UserVotes>546812</UserVotes>
      </Movie>");
```

# Spara XML

- Efter skapat, traverserat, modifierat och transformerat en XML kan den med enkelhet skrivas till en ström eller fil
  - XDocument/XElement .Save(filväg/strömmar/...)

```
XElement movies = new XElement(..);

movies.Save("C:/movies.xml"));

StreamWriter writer = new StreamWriter("C:/movies.xml");
movies.Save(writer);
```