Haladó Fejlesztési Technikák

ÓBUDAI EGYETEM
NEUMANN JÁNOS INFORMATIKAI KAR

Modul 2

Adattárolás XML fájlokban

Adattárolás JSON fájlokban

Nyelvbe ágyazott lekérdezések (LINQ)

LINQ lekérdezések objektumgyűjteményeken

LINQ lekérdezések XML fájlokon

LINQ lekérdezések JSON fájlokon

Adattárolás eddigi módjai

- Gyűjteményeinket általában **txt** fájlokba írtuk ki („szerializáltuk”)
- Pl: **Béla*taxisofőr*55*200000**
  - Jelentése: név, munkakör, életkor, fizetés * jelekkel szeparálva
- Megírtuk rá a saját szerializáló függvényünket

```
public string ToSingleLine()
{
        return $"{Name}*{Job}*{Age}*{Salary}";
}
```

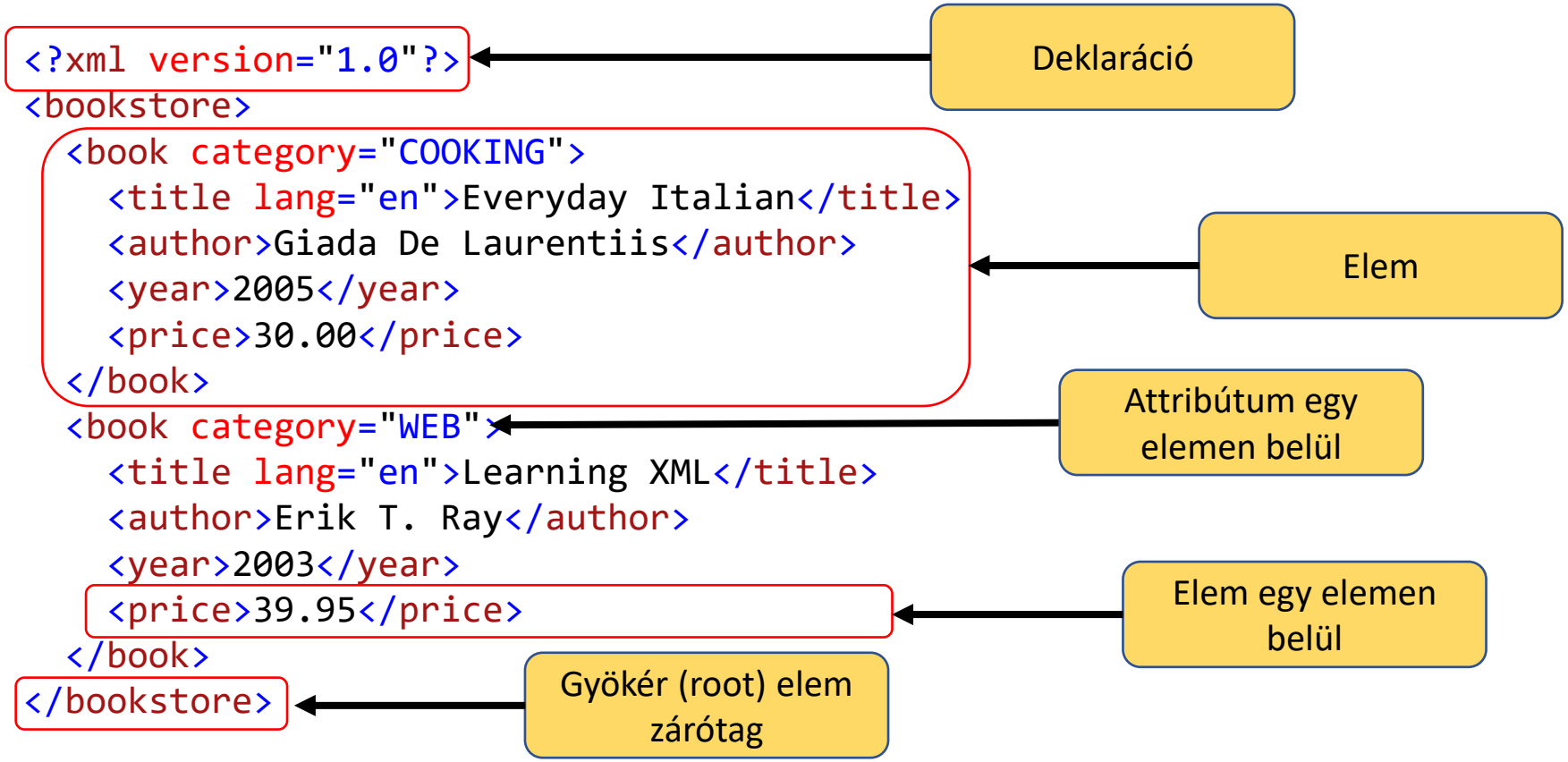
- És a deszerializáló függvényünket is

```
public void FromSingleLine(string line)
{
        string[] pieces = line.Split('*');
        Name = pieces[0];
        Job = pieces[1];
        Age = int.Parse(pieces[2]);
        Salary = int.Parse(pieces[3]);
}
```

Mi a helyzet az egymásba ágyazott objektumokkal (=asszociáció/kompozíció)? Pl. hogyan lehetne eltárolni egy bináris keresőfát? NEHEZEN.

XML formátum

- Hierarchikus adatleíró formátum
- Elemekből (element/node) és attribútumokból (attribute) áll



XML formátum

- Saját objektumainkhoz tervezünk egy XML reprezentációt
- Al-elemek vagy attribútumok? → mindegy!
- Kötelező egyetlen gyökérelem (példában: bookstore)
- Egymásba ágyazás lezárásainak megfelelő sorrendben kell történnie és mindent le kell zárni
- Kisbetű-nagybetű érzékeny
- **Well-formed XML**
  - W3C elvárásoknak megfelelő a formátuma
- **Valid XML**
  - Megadott sémának megfelel a formátuma

XML kezelése

- **XDocument, XElement, XAttribute** osztályok segítségével
- Egy XML-node reprezentációja C# nyelven

```
<year>2003</year>
```

```
XElement yearnode = new XElement("year", 2003);
```

- **XElementnek** van **Name** és **Value** tulajdonsága
- Node-ok egymásbaágyazhatóak az **Add()** metódussal

```
XElement book = new XElement("book");
book.Add(new XElement("title", "Learning XML"));
book.Add(new XElement("author", "Erik T. Ray"));
book.Add(new XElement("year", 2003));
book.Add(new XElement("price", 39.95));
```



```
<book>
   <title>Learning XML</title>
   <author>Erik T. Ray</author>
   <year>2003</year>
   <price>39.95</price>
 </book>
```

XML kezelése

- **XElement** paraméterezése: **XName** name, **object** content
  - **name**: stringként adjuk meg
  - **content**: bármilyen objektum, amin a ToString() overrideolva van
- Van egy **params object[]** túlterhelése is →kb. bármekkora struktúra leírható egy utasítással

- Attribútum hozzáadása egy nodehoz

```
<title lang="en">Learning XML</title>
```

```
XElement title = new XElement("title", "Learning XML");
XAttribute titleattr = new XAttribute("lang", "en");

title.Add(titleattr);
```

XML elmentése


```
XDocument xdoc = new XDocument();

XElement root = new XElement("bookstore");

xdoc.Add(root);


XElement book = new XElement("book");

book.Add(new XElement("title", "Learning XML"));
book.Add(new XElement("author", "Erik T. Ray"));
book.Add(new XElement("year", 2003));
book.Add(new XElement("price", 39.95));

root.Add(book);

xdoc.Save("data.xml");
```

XML fájlt reprezentáló objektum létrehozása

Kötelező gyökérelem elkészítése

Kötelező gyökérelem dokumentumhoz adása

Egy al-elem elkészítése

Al-elem feltöltése al-elemekkel

Al-elem gyökérelem alá szúrása

Dokumentum elmentése

Feladat

- Kérjünk be a felhasználótól N darab filmet!
- A filmnek legyen címe, hossza és megjelenésének éve!
- Minden konzolbekérés után kérdezzük meg, hogy szeretne-e még új rekordot felvinni!

- A bekért filmeket mentsük el fájlba!
  - Mentsük el txt-be a hagyományos módon!
  - Mentsük el xml-be az imént tanult módon!

XML bejárása

- **XDocument** létrehozásának 3 módja van
  - new **XDocument()** ← üres objektum, feltöltjük és kimentjük (lásd: előzőek)
  - **XDocument.Load**(**string** uri) ← fájl elérési út vagy URL cím alapján betölt egy XML fájt egy Xdocument példányba és a példányt visszaadja
  - **XDocument.Parse**(**string** text) ← hogyha az xml tartalom egy string változóba be van töltve, akkor abból elkészíti a feltöltött XDocument példányt (ritkán használjuk)

XML bejárása

- **XDocument** és **XElement** rendelkeznek az alábbi metódusokkal:
  - **Element**(**XName** name) → visszaadja az ilyen nevű al-elementek közül az elsőt (**XElement**)
  - **Elements**() → visszaadja az összes al-elementet (**IEnumerable<XElement>**)
  - **Elements**(**XName** name) → visszaadja az ilyen nevű összes al-elementet (**IEnumerable<XElement>**)
  - **Descendants**(**Xname** name) → visszaadja az ilyen nevű összes al-elementet (**IEnumerable<XElement>**) rekurzívan!
  - **Attribute**(**Xname** name) → visszaadja az ilyen nevű attribútumok közül az elsőt (**XAttribute**)
  - **Attributes**(**Xname** name) →visszaadja az ilyen nevű attr-ok közül az összeset (**IEnumerable<XAttribute>**)

```
XDocument doc = XDocument.Load("data.xml");
IEnumerable<XElement> books = doc.Element("bookstore").Elements("book");
foreach (XElement item in books)
{
        Console.WriteLine(item.Element("title").Value);
        Console.WriteLine(item.Element("author").Value);
}
```

XML szerializáció

- Automatikus XML kezelés is rendelkezésünkre áll

szerializáció

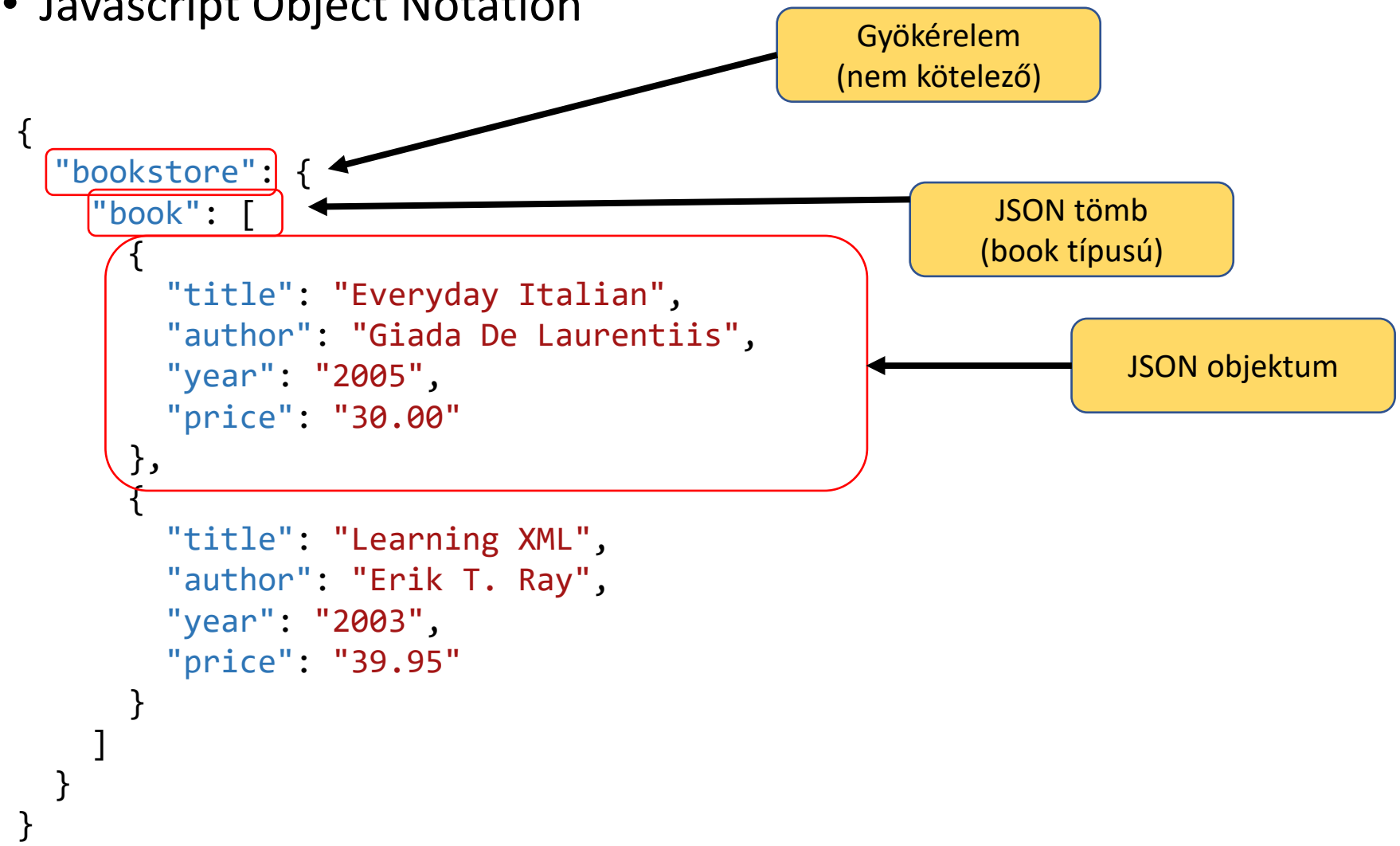
```
Person p = new Person()
{
        Name = "Peter",
        Job = "Spider-man"
};

XmlSerializer writer =
        new XmlSerializer(typeof(Person));

FileStream file = File.Create("data.xml");
writer.Serialize(file, p);
file.Close();
```

deszerializáció

```
XmlSerializer reader =
        new XmlSerializer(typeof(Person));
StreamReader file = new StreamReader("data.xml");
Person p = (Person)reader.Deserialize(file);
file.Close();
```

JSON formátum

- Javascript Object Notation




```
{
  "bookstore": {
    "book": [
      {
        "title": "Everyday Italian",
        "author": "Giada De Laurentiis",
        "year": "2005",
        "price": "30.00"
      },
      {
        "title": "Learning XML",
        "author": "Erik T. Ray",
        "year": "2003",
        "price": "39.95"
      }
    ]
  }
}
```

JSON különbségek

- **XML-től eltérően**:
  - Nem kötelező gyökérelem
  - Objektum-orientált megközelítéshez jobban passzol szerkezetre
  - Nincs zárótag
  - Tömböket is lehet benne használni

- **.NET-ben használata**:
  - **Newtonsoft.JSON** libraryvel
  - Microsoft is ma már ezt a külső libet használja
  - NUGET csomagkezelővel telepíthető

JSON szerializáció

- Automatikus JSON kezelés (Newtonsoft.JSON)

szerializáció

```
Person p = new Person()
{
        Name = "Peter",
        Job = "Spider-man"
};

string json =
        JsonConvert.SerializeObject(p);
File.WriteAllText("data.json", json);
```

deszerializáció

```
string json =
        File.ReadAllText("data.json");
Person p =
        JsonConvert.DeserializeObject<Person>(json);
```

LINQ

- Language Integrated Queries
- Gyűjtemények **forrás** és **struktúra** független kezelése egyszerűen
  - Tanult programozási tételek beépítve
  - Bármilyen adatforráson (tömb, lista, XML, adatbázis) ugyanúgy
- Mit jelent ez a gyakorlatban?
  - „Kérjük le a 60 évnél fiatalabb felhasználókat úgy, hogy nem tudjuk mi az adatforrás”
- Mit kell a használatához ismerni?
  - ✓ Lambda kifejezéseket
  - Névtelen osztályokat és a var kulcsszót
  - Kiegészítő metódusokat (extension method)
  - LINQ operátorokat (=kiegészítő metódusok)

Var kulcsszó

- Egy értékadás operátor bal oldalára ki szoktuk írni a típust

```
Person peter = new Person();
```

- De igazából ez nem is annyira fontos, mert a fordító pontosan tudja, hogy az operátor jobb oldaláról milyen típusú adat érkezik, ezért egyszerűsíthetünk

```
var peter = new Person();
```

- A fordító határozza meg, hogy a peter objektum milyen típusú
- Kötelező azonnal értéket, adni neki. Ilyen **nem** lehet:

```
var peter;
```

- Metódus visszatérési értékben sem állhat

Ez nem egy általános típus! Csupán majd a fordító behelyettesíti a valódi típust.

Névtelen osztályok

- Lehetőség van a névtelen metódusokhoz hasonlóan egyszer használatos osztályt létrehozni

```
var peter = new
{
        RealName = "Peter Paker",
        HeroName = "Spider-Man"
};
```

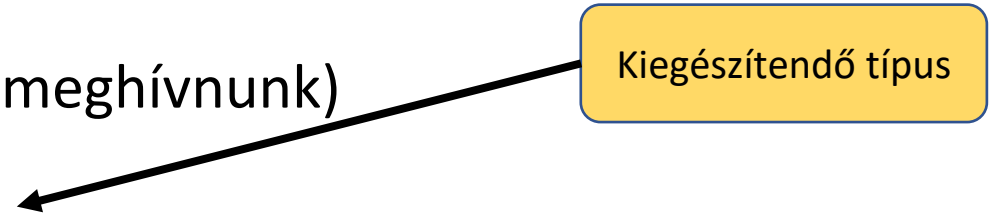
- Peter típusa ekkor anonymus
- Ez a fajta szintaxis (**object initializer**) működik rendes osztályoknál is, de ott opcionális

```
Person peter = new Person()
{
        Name = "Peter Parker",
        Job = "Superhero"
};
```

- Tulajdonságoknak tudunk gyorsan kezdőértéket adni

Kiegészítő metódusok

- Egy meglévő típushoz futásidőben új metódusok „ragasztása”
- Pl: jó lenne ha a DateTime típus egyéb formában is vissza tudná adni a dátumot (és ez saját funkciójaként jelenne meg)
- Készítünk egy statikus osztályt (sehol nem kell meghívnunk)

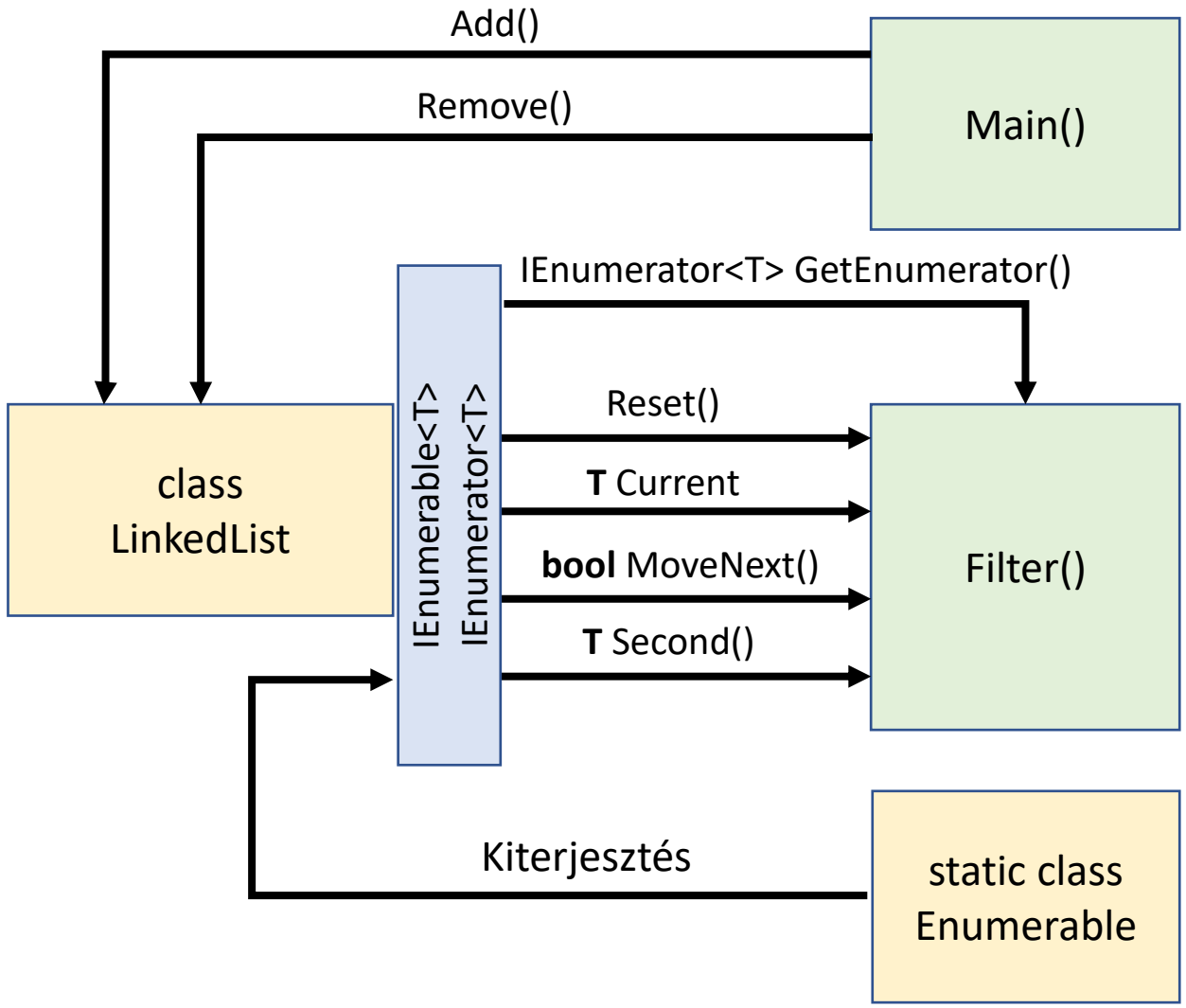



```
public static class DateTimeExtender
{
        public static string ToCustomFormat(this DateTime source)
        {
                return $"{source.Year}:{source.Month}:{source.Day}";
        }
}
```

- És már tudjuk is használni a kódban bárhol

```
Console.WriteLine(DateTime.Now.ToCustomFormat());
```

Hogy működik a LINQ?



- A LinkedList helyett bármilyen osztály szerepelhet, ami **IEnumerable<T>**-t megvalósít
- Ekkor van neki: Reset(), Current, MoveNext()

```
public static class Enumerable
{
    public static T Second<T>(this IEnumerable<T> source)
    {
        IEnumerator<T> enumerator = source.GetEnumerator();
        if (enumerator.MoveNext())
        {
            if (enumerator.MoveNext())
            {
                return enumerator.Current;
            }
        }
        throw new ArgumentException("Less than 2 items...");
    }
}
```

Feladat

- Készítsünk egy olyan **IEnumerable<T>** extension methodot, ami bármilyen gyűjteményből megadja a
  - Középső elemet!
  - Legnagyobb elemet!
  - Adott kritériumoknak megfelelő elemeket!

Beépített LINQ metódusok

- Két gyűjtemény egymás után fűzése (nem halmazok!)

```
var allNumbers = first.Concat(second);
```

- Elem létezésének vizsgálata

```
bool doesContainFour = first.Contains(4);
```

- Ismétlődések kivágása (halmazzá alakítás)

```
var onlyDifferentNumbers = first.Distinct();
```

- Halmaz metszet

```
var sameItems = first.Intersect(second);
```

- Halmaz unió

```
var unionOfSets = first.Union(second);
```

- Halmaz különbség

```
var diffOfSets = first.Except(second);
```

Beépített LINQ metódusok

- Sorrendezés → rendezett másolatot ad vissza IOrderedEnumerable<T> interfésszel

```
var items = first.OrderBy(t => t); //t önmagában IComparable
var items = students.OrderBy(t => t.Age); //tulajdonság kijelölés
```

- Csökkenő sorrend

```
var items = students.OrderByDescending(t => t.Age); //tulajdonság kijelölés
```

- Szűrés / kiválogatás →IEnumerable<T> visszatérés

```
var filtered = students.Where(t => t.Age > 18);
```

- Megszámlálás →int visszatérés

```
int counter = students.Count(t => t.Age > 18);
```

Beépített LINQ metódusok

- Minden elemre igaz-e az állítás?

```
bool statement = first.All(t => t % 2 == 0);
```

- Van-e elem a gyűjteményben?

```
bool isElements = numbers.Any();
```

- 1 elemű gyűjteménnyel visszatérés, ha a gyűjtemény üres (paraméter: mi legyen az 1 elem)

```
var elements = numbers.DefaultIfEmpty(0);
```

- Adott sorszámú elem visszaadása

```
var element = numbers.ElementAt(5);
```

- Az első n db elem visszaadása

```
var elements = numbers.Take(5);
```

- Az első n db elem kihagyása

```
var elements = numbers.Skip(3);
```

Beépített LINQ metódusok

- Az első T tulajdonságú elem megkeresése (ha nincs → Exception)

```
var element = numbers.First(t => t % 7 == 0);
```

- Az első T tulajdonságú elem megkeresése (ha nincs → null)

```
var element = numbers.FirstOrDefault(t => t % 7 == 0);
```

- Az egyetlen T tulajdonságú elem megkeresése (ha nincs vagy több is van →Exception)

```
var element = numbers.Single(t => t % 7 == 0);
```

- Az egyetlen T tulajdonságú elem megkeresése (ha nincs → null, ha több van → Exception)

```
var element = numbers.SingleOrDefault(t => t % 7 == 0);
```

Tulajdonság kiválasztás

- Talán az egyik leghasznosabb LINQ metódus
- T típusú gyűjteményt bármilyen gyűjteményre alakíthatunk transzformáló logikával
- Hogyan oldanánk meg, hogy **Student** lista helyett csak a hallgatók neve legyen stringként?

```
List<string> studentNames = new List<string>();

foreach (var student in students)
{
        studentNames.Add(student.Name);
}
```

- LINQ **Select** metódus erre való

```
var studentNames = students.Select(t => t.Name);
```

A Select kimenete a „t” lambda operandus egy megjelölt tulajdonsága vagy akár egy új típus

Tulajdonság kiválasztás

- LINQ **Select** metódus tehát képes csak egy tulajdonság alapján új gyűjteményt csinálni

```
var studentNames = students.Select(t => t.Name);
```

- De visszatérhetünk itt egy másik típussal is (pl. Student → Worker konvertáló)

```
var workers = students
        .Select(t => new Worker(t.Name, t.Age, t.Name.Length * t.Age));
```

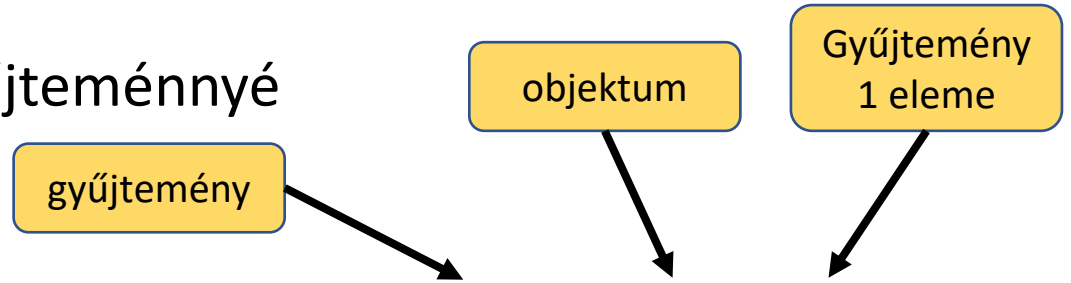
- De visszatérhetünk egy névtelen típussal is

```
var result = students.Select(t => new
{
        StudentName = t.Name,
        BirthYear = DateTime.Now.Year - t.Age
});
```

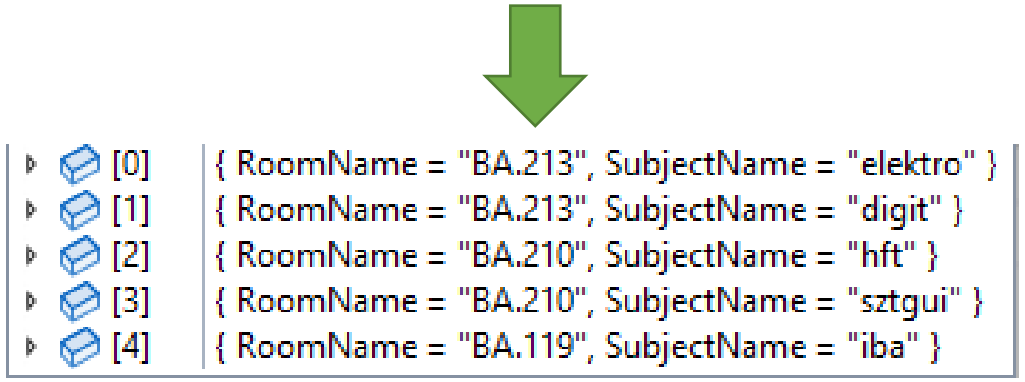
SelectMany

- Egy összetett objektum simítható ki egy szimpla gyűjteménnyé

```
var rooms = new Room[]
{
        new Room("BA.213", new Subject[]
        {
                new Subject("elektro"),
                new Subject("digit")
        }),
        new Room("BA.210", new Subject[]
        {
                new Subject("hft"),
                new Subject("sztgui")
        }),
        new Room("BA.119", new Subject[]
        {
                new Subject("iba")
        }),
};
```




```
var subjectList = rooms.SelectMany(t => t.Subjects, (room, subject) => new
{
        RoomName = room.RoomName,
        SubjectName = subject.SubjectName
});
```



LINQ láncolás

- A legtöbb LINQ metódus valamilyen gyűjteményen interfésszel tér vissza (általában **IEnumerable<T>**)
- De ez lehet egy újabb LINQ metódus bemenete is → láncolhatóság lehetővé válik

```
var result = students
        .Where(t => t.Age > 20)
        .OrderBy(t => t.Name)
        .Reverse()
        .Select(t => t.Name.ToUpper());
```

METHOD SYNTAX

- Használhatunk helyette deklaratív megközelítést (SQL-szerű → de nem SQL)

```
var result = from t in students
             where t.Age > 20
             orderby t.Name descending
             select t.Name.ToUpper();
```

QUERY SYNTAX

Feladat

- Készítsünk egy olyan programot, amely filmek gyűjteményét beolvassa **JSON** fájlból, majd
  - Leszűri a 2006-os filmeket (**Where** metódussal)
  - Minden film címét és rendezőjének nevét **XElement**-re alakítja! (**Select** metódussal)
  - Kimenti XML-be a címüket és a rendező nevét **XDocument** módszerrel!

- Forrás JSON
  - https://nikprog.hu/samples/movie.json

Aggregáló metódusok

- Tipikusan gyűjtemény → egy db mérőszám átalakítás
- Összegzés (sorozatszámítás)

```
int sum = numbers.Sum();
```

- Átlagolás

```
double avg = numbers.Average();
```

- Lehet ez komplexebb is

```
var avgJuniorAge = students
        .Where(t => t.Age < 18)
        .Select(t => t.Age)
        .Average();

var avgJuniorAge = students.Where(t => t.Age < 18).Average(t => t.Age);
```

Csoportosítás

- Egy gyűjtemény széttördelése több gyűjteményre
- Valamilyen kategorizálható változó szerint
- Kategoriális változóra példák (enum-szerű)
  - Tapasztalat: junior | medior | senior
  - Munka-sáv: délelőtt | délután | éjjel
  - Tantárgy számonkérés: vizsga | évközi jegy

Név	Fizetés	Életkor	Tapasztalat
Béla	350e	31	Medior
Géza	250e	25	Junior
József	320e	35	Junior
Péter	450e	32	Senior
Katalin	400e	27	Medior
Ágnes	450e	40	Senior
Zsuzsanna	250e	25	Junior

Group key

Név	Fizetés	Életkor	Tapasztalat
Géza	250e	25	Junior
József	320e	35	Junior
Zsuzsanna	250e	25	Junior

Név	Fizetés	Életkor	Tapasztalat
Béla	350e	31	Medior
Katalin	400e	27	Medior

Név	Fizetés	Életkor	Tapasztalat
Péter	450e	32	Senior
Ágnes	450e	40	Senior

Csoportosítás

- A csoportosító kód

```
var groups = workers.GroupBy(t => t.Level);
```

- Ennek eredménye:
  - **IEnumerable<IGrouping<string, Worker>>**
- Általánosan:
  - **IEnumerable<IGrouping<TKey, TElement>>**
- Gyakorlatban a példán keresztül
  - Három elemű gyűjtemény
  - Minden elem önmagában egy bejárható Worker gyűjtemény
  - Minden Worker gyűjteménynek van egy string kulcsa: junior | medior | senior

Név	Fizetés	Életkor	Tapasztalat
Géza	250e	25	Junior
József	320e	35	Junior
Zsuzsanna	250e	25	Junior

Név	Fizetés	Életkor	Tapasztalat
Béla	350e	31	Medior
Katalin	400e	27	Medior

Név	Fizetés	Életkor	Tapasztalat
Péter	450e	32	Senior
Ágnes	450e	40	Senior

Csoportosítás

- A csoportosító kód

```
var groups = workers.GroupBy(t => t.Level);
```

- A csoportok kiírása szépen elkülönítve a konzolon

```
foreach (var item in groups)
{
        Console.WriteLine(item.Key + " level programmers: ");
        Console.WriteLine();
        foreach (var programmer in item)
        {
                Console.WriteLine($"{programmer.Name}");
        }
        Console.WriteLine("------------------------------");
}
```

Név	Fizetés	Életkor	Tapasztalat
Géza	250e	25	Junior
József	320e	35	Junior
Zsuzsanna	250e	25	Junior

Név	Fizetés	Életkor	Tapasztalat
Béla	350e	31	Medior
Katalin	400e	27	Medior

Név	Fizetés	Életkor	Tapasztalat
Péter	450e	32	Senior
Ágnes	450e	40	Senior

- Az egyes csoportokon aggregáló metódusokat is futtathatunk

```
Console.WriteLine(item.Key + " level programmers: ");
Console.WriteLine($"Average age: {item.Average(t => t.Age)}");
Console.WriteLine();
```

Új kód

Feladat

- Készítsünk egy olyan programot, amely filmek gyűjteményét beolvassa **JSON** fájlból, majd
  - Kategorizálja őket műfaj szerint (**GroupBy**)
  - Konzolos menüt jelenít meg az egyes műfajokra
  - Az adott menüben látjuk
    - Az ide tartozó filmeket, rendezőjüket
    - Az összes ide tartozó film darabszámát
- A feladathoz használjuk fel a **ConsoleMenu-simple** nuget csomagot!
- Legyen lehetőség egy online URL-ről kérni a filmeket!

Kimutatás készítés

- Általában azért készítünk kimutatásokat, hogy összehasonlítsunk csoportokat
- Általában a csoportokon belüli egyes elemekre nincs is nagyon szükségünk, csak aggregált eredményekre
- Példák:
  - Tapasztalati szintenkénti átlagfizetés
  - Tapasztalati szintenkénti átlagéletkor
  - Korcsoportonkénti átlagfizetés
    - Pl: 20-30 | 30-40 | 40-50 (kategoriális így már!)
- Tapasztalati szintenkénti átlagfizetés

Név	Fizetés	Életkor	Tapasztalat
Béla	350e	31	Medior
Géza	250e	25	Junior
József	320e	35	Junior
Péter	450e	32	Senior
Katalin	400e	27	Medior
Ágnes	450e	40	Senior
Zsuzsanna	250e	25	Junior

```
var groups = workers.GroupBy(t => t.Level);
foreach (var item in groups)
{
        Console.WriteLine($"{item.Key}: {item.Average(t => t.Salary)}");
}
```

Kimutatás készítés

- Tapasztalati szintenkénti átlagfizetés

```
var groups = workers.GroupBy(t => t.Level);
foreach (var item in groups)
{
        Console.WriteLine($"{item.Key}: {item.Average(t => t.Salary)}");
}
```

- Megfelelő típus készítése a szint – átlagfizetés tárolására

```
public class AvgSalaryOfLevels
{
        public string LevelName { get; set; }
        public double AvgSalary { get; set; }
}
```

- Típusos gyűjtemény lekérése

```
var result = workers.GroupBy(t => t.Level).Select(t => new AvgSalaryOfLevels()
{
        LevelName = t.Key,
        AvgSalary = t.Average(z => z.Salary)
});
```

Kimutatás készítés

- Névtelen típusba kigyűjtés

```
var result = workers.GroupBy(t => t.Level).Select(t => new
{
        LevelName = t.Key,
        AvgSalary = t.Average(z => z.Salary)
});
```

- Ugyanez query syntax-szal (általában egyszerűbb)

```
var result = from x in workers
             group x by x.Level into g
             select new
             {
                  LevelName = g.Key,
                  AvgSalary = g.Average(z => z.Salary)
             };
```

JOIN - összefűzés

- INNER JOIN: Tipikusan két gyűjteményből szeretnénk egyet csinálni
- És ezeket valamilyen közös mező mentén fűzzük össze (pl: szervezeti egység)

Név	Fizetés	Sz. Egység
Béla	350e	Research
Géza	250e	Development
József	320e	HR
Péter	450e	Development
Katalin	400e	null
Ágnes	450e	HR
Zsuzsanna	250e	Development

Sz. egység	Helyszín	Vezető
Research	Budapest	Béla
Development	Miskolc	Péter
HR	Szeged	Ágnes
Quality	Miskolc	Béla

```
var merged = from w in workers
             join d in departments
             on w.Department equals d.Deptname
             select new
             {
                  WorkerName = w.Name,
                  Location = d.Location
             };
```

Név	Fizetés	Sz. Egység	Helyszín	Vezető
Béla	350e	Research	Budapest	Béla
Géza	250e	Development	Miskolc	Péter
József	320e	HR	Szeged	Ágnes
Péter	450e	Development	Miskolc	Péter
Ágnes	450e	HR	Szeged	Ágnes
Zsuzsanna	250e	Development	Miskolc	Péter

JOIN - összefűzés

- LEFT OUTER JOIN: szeretnénk Katalint-t is az eredményben látni
- És szeretnénk látni, hogy neki nincs szervezeti egysége

Név	Fizetés	Sz. Egység
Béla	350e	Research
Géza	250e	Development
József	320e	HR
Péter	450e	Development
Katalin	400e	null
Ágnes	450e	HR
Zsuzsanna	250e	Development

Sz. egység	Helyszín	Vezető
Research	Budapest	Béla
Development	Miskolc	Péter
HR	Szeged	Ágnes
Quality	Miskolc	Béla

```
var merged = from w in workers
             join d in departments
             on w.Department equals d.Name into sub
             from m in sub.DefaultIfEmpty()
             select new
             {
                      WorkerName = w.Name,
                      Location = m?.Location
             };
```

Név	Fizetés	Sz. Egység	Helyszín	Vezető
Béla	350e	Research	Budapest	Béla
Géza	250e	Development	Miskolc	Péter
József	320e	HR	Szeged	Ágnes
Péter	450e	Development	Miskolc	Péter
Katalin	400e	null	null	null
Ágnes	450e	HR	Szeged	Ágnes
Zsuzsanna	250e	Development	Miskolc	Péter

Feladat

- Fűzzünk össze két adatforrást!
  - https://nikprog.hu/samples/devtask.json
  - https://nikprog.hu/samples/person.xml
- A **person**-ben a dolgozóink szerepelnek, a **devtask**-ban a vállalt feladataik!
- Válaszoljunk az alábbi kérdésre:
  - Adott tudásszintű emberek átlagosan hány feladatot vállalnak?

LINQ to XML, JSON LINQ

- Az X* osztályoknak erős a LINQ támogatása → XML nodeokon közvetlen LINQ lehetőség
- Gyors, egyszerű, nem kell osztályt létrehozni hozzá, deszerializálni, stb.
- Összetett objektumoknál (asszociáció, kompozíció) már körülményes

```
XDocument xDoc = XDocument.Load("data.xml");
var q = from node in xDoc.Descendants("person")
        where node.Element("name").Value.StartsWith("J")
        select node.Value;
```

- JSON is LINQ-zható közvetlenül

```
JArray arr = JArray.Parse(jsonContent);
var query = from x in arr
            where x.ToObject<Person>().Age > 28
            select x;
```

Köszönöm a figyelmet!

Kérdés esetén e-mailben szívesen állok rendelkezésre.