



Scott Allen

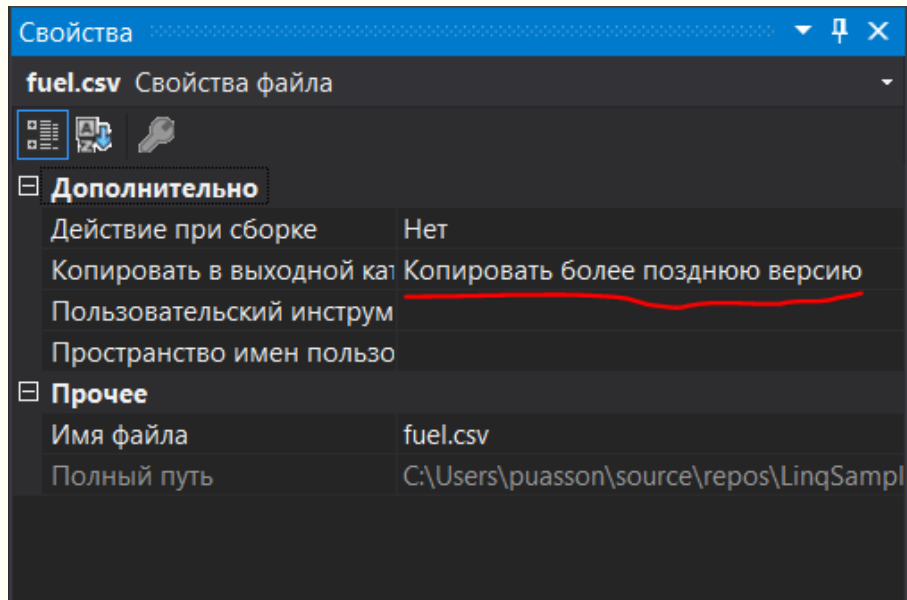
OdeToCode.com - @OdeToCode

ФІЛЬТРУВАННЯ, ВПОРЯДКУВАННЯ ТА ПРОЄКТУВАННЯ ДАНИХ

Питання 8.2.

Розглянемо новий проєкт «Cars»

- Джерелом даних у нього буде файл fuel.csv.
 - Він містить понад 1200 записів про автомобілі, випущені в 2016 році та включає рейтинги щодо ефективності використання ними палива в місті (City FE), на шосе (Hwy FE) та в комбінованому сценарії (Comb FE).
 - Джерело даних.



```
1 Model Year,Division,Carline,Eng Displ,# Cyl,City FE,Hwy FE,Comb FE
2 2016,ALFA ROMEO,4C,1.8,4,24,34,28
3 2016,Aston Martin Lagonda Ltd,V12 Vantage S,6.0,12,12,18,14
4 2016,Aston Martin Lagonda Ltd,V8 Vantage,4.7,8,14,21,16
5 2016,Aston Martin Lagonda Ltd,V8 Vantage,4.7,8,13,19,15
6 2016,Aston Martin Lagonda Ltd,V8 Vantage S,4.7,8,14,21,16
7 2016,Aston Martin Lagonda Ltd,V8 Vantage S,4.7,8,13,19,15
8 2016,Aston Martin Lagonda Ltd,Vantage GT,4.7,8,14,21,16
9 2016,Aston Martin Lagonda Ltd,Vantage GT,4.7,8,13,19,15
10 2016,Audi,TT Roadster quattro,2.0,4,23,30,26
11 2016,BMW,M4 GTS,3.0,6,16,23,19
12 2016,BMW,Z4 sDrive28i,2.0,4,22,33,26
13 2016,BMW,Z4 sDrive28i,2.0,4,22,34,26
14 2016,BMW,Z4 sDrive35i,3.0,6,17,24,20
15 2016,BMW,Z4 sDrive35is,3.0,6,17,24,20
16 2016,Chevrolet,CORVETTE,6.2,8,13,23,16
17 2016,Chevrolet,CORVETTE,6.2,8,16,29,20
18 2016,Chevrolet,CORVETTE,6.2,8,15,22,18
19 2016,Chevrolet,CORVETTE,6.2,8,17,29,21
20 2016,Dodge,Viper SRT,8.4,10,12,21,15
```

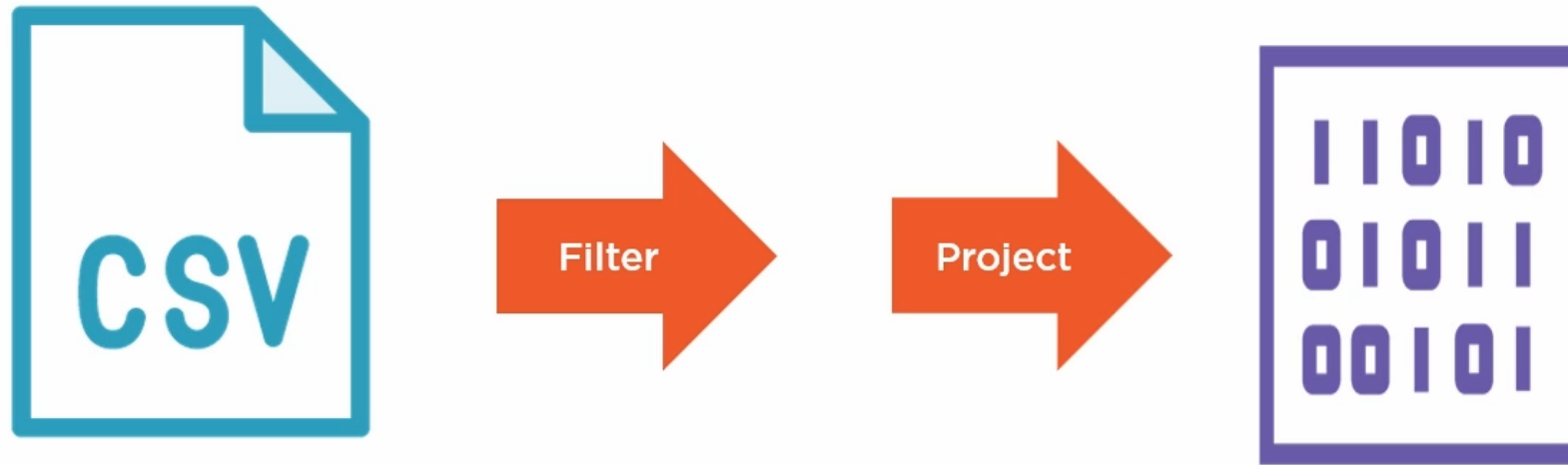
Розглянемо новий проєкт «Cars»

- Цікаве питання до джерела даних: які автомобілі мають найкращу ефективність використання палива? (електрокарів у джерелі даних немає).
- Перенесемо дані з файлу в колекцію об'єктів типу Car:

```
public class Car
{
    public int Year { get; set; }
    public string Manufacturer { get; set; }
    public string Name { get; set; }
    public double Displacement { get; set; }
    public int Cylinders { get; set; }
    public int City { get; set; }
    public int Highway { get; set; }
    public int Combined { get; set; }
}
```

- Послідовність дій:
 - Відкрити csv-файл та зчитати його в масив рядків.
 - Деякі рядки обробляти не потрібно (заголовки таблиці, порожній рядок у кінці файлу) – їх відфільтруємо.
 - Далі перетворюємо (проєктуємо) рядок у Car-об'єкт за допомогою оператора Select().
 - Перетворюємо дані з файлу в in-memory колекцію Car-об'єктів, з якою будемо працювати.

Потокові (streaming) оператори LINQ



■ 1) Обробка даних з файлу:

- Зчитуємо всі рядки з файлу, пропускаючи перший рядок – заголовок таблиці.
- Відфільтровуємо останній порожній рядок, вказуючи на наявність більш, ніж одного елемента в звичайному рядку.
- Доповнимо клас Car простою реалізацією розбору csv-файлу – методом ParseFromCsvs().
- Здійснимо негайне виконання (оператор ToList()), щоб не звертатись до файлу при зчитуванні кожного рядка.

```
private static List<Car> ProcessFile(string path)
{
    return
        File.ReadAllLines(path)
            .Skip(1)
            .Where(line => line.Length > 1)
            .Select(Car.ParseFromCsv)
            .ToList();
}
```

Оновлений вигляд класу Car

```
public class Car
{
    public int Year { get; set; }
    public string Manufacturer { get; set; }
    public string Name { get; set; }
    public double Displacement { get; set; }
    public int Cylinders { get; set; }
    public int City { get; set; }
    public int Highway { get; set; }
    public int Combined { get; set; }


    internal static Car ParseFromCsv(string line)
    {
        var columns = line.Split(',');

        return new Car
        {
            Year = int.Parse(columns[0]),
            Manufacturer = columns[1],
            Name = columns[2],
            Displacement = double.Parse(columns[3], _nfi),
            Cylinders = int.Parse(columns[4]),
            City = int.Parse(columns[5]),
            Highway = int.Parse(columns[6]),
            Combined = int.Parse(columns[7])
        };
    }
}
```

- Не вводимо додаткових перевірок, щоб не ускладнювати розбір теми.

```
static void Main(string[] args)
{
    var cars = ProcessFile("fuel.csv");
    foreach (var car in cars)
    {
        Console.WriteLine(car.Name);
    }
}

private static List<Car> ProcessFile(string path)
{
    return
        File.ReadAllLines(path)
            .Skip(1)
            .Where(line => line.Length > 1)
            .Select(Car.ParseFromCsv)
            .ToList();
}
```



```
C:\Windows\system32\cmd.exe
AMG GLE 63 S (coupe)
G 550
GL 350 BLUETEC 4MATIC
GL 450 4MATIC
GL 550 4MATIC
GLE 300 d 4MATIC
GLE 350 4MATIC
GLE 350 4MATIC
GLE 350 d 4MATIC
GLE 400 4MATIC
GLE 450 AMG (coupe)
Cayenne
Cayenne Diesel
Cayenne GTS
Cayenne S
Cayenne Turbo
Cayenne Turbo S
4RUNNER 4WD
4RUNNER 4WD
HIGHLANDER AWD
HIGHLANDER HYBRID AWD
HIGHLANDER HYBRID AWD LE Plus
LAND CRUISER WAGON 4WD
SEQUOIA 4WD
SEQUOIA 4WD FFV
Touareg
Touareg
XC90 AWD
XC90 AWD
Press any key to continue . . .
```

Використання синтаксису запитів

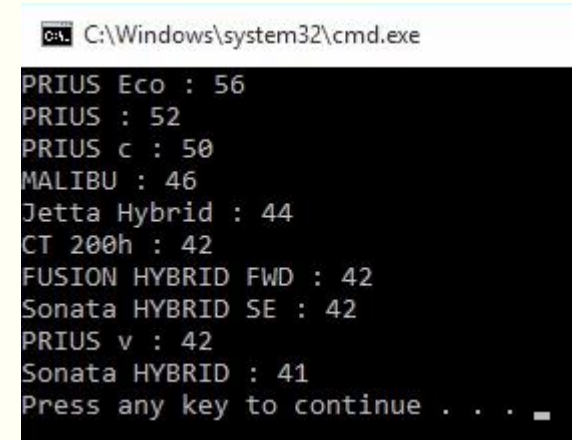
```
private static List<Car> ProcessFile(string path)
{
    var query =
        from line in File.ReadAllLines(path).Skip(1)
        where line.Length > 1
        select Car.ParseFromCsv(line);

    return query.ToList();
}
```

- У синтаксисі запитів немає оператора ToList(), тому доводиться застосовувати його окремо.
 - Далі шукатимемо найбільш ефективні з точки зору використання палива автомобілі.

Найбільш ефективні автомобілі

```
var cars = ProcessFile("fuel.csv");  
var query = cars.OrderByDescending(c => c.Combined);  
  
foreach (var car in query.Take(10))  
{  
    Console.WriteLine($"{car.Name} : {car.Combined}");  
}
```



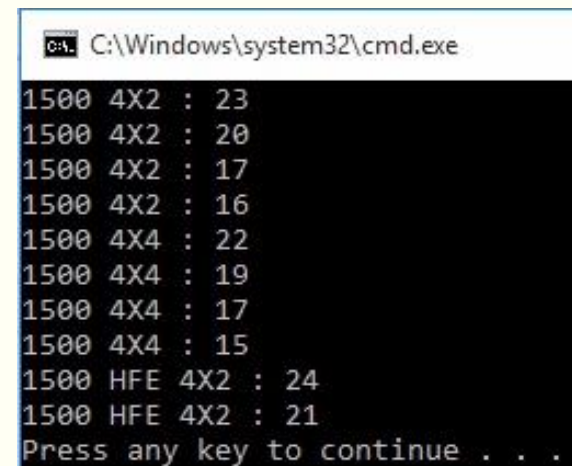
```
C:\Windows\system32\cmd.exe  
PRIUS Eco : 56  
PRIUS : 52  
PRIUS c : 50  
MALIBU : 46  
Jetta Hybrid : 44  
CT 200h : 42  
FUSION HYBRID FWD : 42  
Sonata HYBRID SE : 42  
PRIUS v : 42  
Sonata HYBRID : 41  
Press any key to continue . . .
```

- Велика кількість автомобілів із показником 42 може спричинити потребу в додаткових критеріях сортування.

- Некоректний підхід – використовувати ще один OrderBy.

```
var query = cars.OrderByDescending(c => c.Combined)  
                .OrderBy(c => c.Name);
```

- LINQ бачить потребу в окремому сортуванні за назвою та ефективно пересортує дані, відмінивши попередні напрацювання



```
C:\Windows\system32\cmd.exe  
1500 4X2 : 23  
1500 4X2 : 20  
1500 4X2 : 17  
1500 4X2 : 16  
1500 4X4 : 22  
1500 4X4 : 19  
1500 4X4 : 17  
1500 4X4 : 15  
1500 HFE 4X2 : 24  
1500 HFE 4X2 : 21  
Press any key to continue . . .
```

Кілька критеріїв сортування

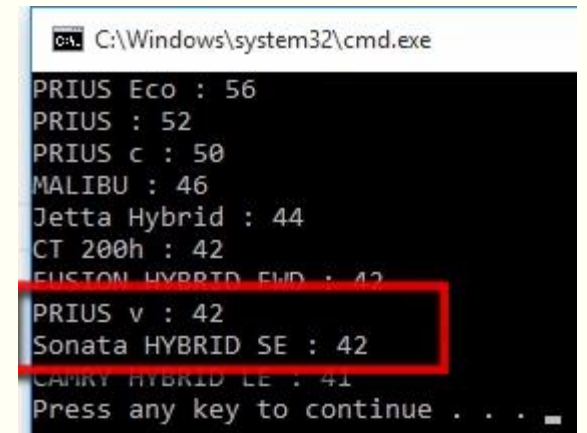
- Використовуються оператори `ThenBy()` / `ThenByDescending()`

```
var query = cars.OrderByDescending(c => c.Combined)
                  .ThenBy(c => c.Name);
```

- Той же синтаксис, записаний за допомогою синтаксису запитів:

```
var query =
    from car in cars
    orderby car.Combined descending, car.Name ascending
    select car;
```

- Ключове слово `ascending` визначає значення за умовчанням, тому його можна опускати.
- Оператор `select` виконує відображення (проєкцію) даних у відповідності до критеріїв вибірки.
- Допускається також реверс послідовності елементів (оператор `Reverse()`), проте він недоступний в синтаксисі запитів.

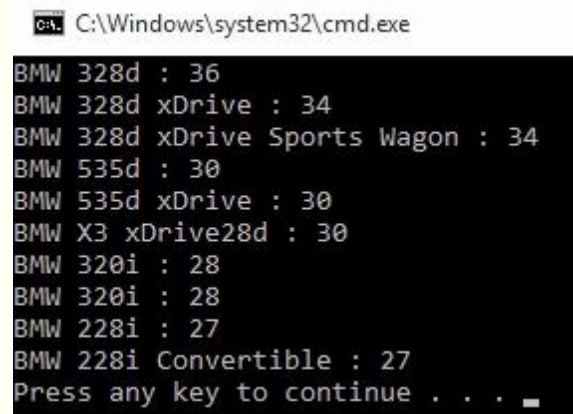


```
C:\Windows\system32\cmd.exe
PRIUS Eco : 56
PRIUS : 52
PRIUS c : 50
MALIBU : 46
Jetta Hybrid : 44
CT 200h : 42
ELUSTON HYBRID FWD : 42
PRIUS v : 42
Sonata HYBRID SE : 42
CAMRY HYBRID LE : 41
Press any key to continue . . .
```


Фільтрування даних за допомогою Where() та First()

- Спробуємо знайти найефективніші автомобілі з точки зору витрати палива, проте враховуючи виробника (тут візьмемо BMW) та рік випуску (тут – 2016).

```
var query =  
    from car in cars  
    where car.Manufacturer == "BMW" && car.Year == 2016  
    orderby car.Combined descending, car.Name ascending  
    select car;
```



```
C:\Windows\system32\cmd.exe  
BMW 328d : 36  
BMW 328d xDrive : 34  
BMW 328d xDrive Sports Wagon : 34  
BMW 535d : 30  
BMW 535d xDrive : 30  
BMW X3 xDrive28d : 30  
BMW 320i : 28  
BMW 320i : 28  
BMW 228i : 27  
BMW 228i Convertible : 27  
Press any key to continue . . .
```

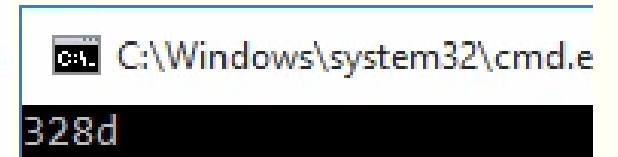
- Аналогічний синтаксис, записаний за допомогою методів розширення.
 - Select() тут не обов'язковий, а просто показаний для порівняння.
 - Результат буде той же.

```
var query2 =  
    cars.Where(c => c.Manufacturer == "BMW" && c.Year == 2016)  
        .OrderByDescending(c => c.Combined)  
        .ThenBy(c => c.Name)  
        .Select(c => c);  
  
foreach (var car in query2.Take(10))  
{  
    Console.WriteLine($"{car.Manufacturer} {car.Name} : {car.Combined}");  
}
```

Фільтрування даних за допомогою Where() та First()

- Визначимо найбільш ефективний автомобіль з перелічених:

```
var top =  
    cars.Where(c => c.Manufacturer == "BMW" && c.Year == 2016)  
        .OrderByDescending(c => c.Combined)  
        .ThenBy(c => c.Name)  
        .Select(c => c)  
        .First();  
  
Console.WriteLine(top.Name);
```



- Замість повернення послідовності автомобілів оператор First() повертає лише один автомобіль.
- Тому оператор не передбачає відкладеного виконання та ітерування по результатах роботи.
- Метод First() передбачає перевантажену реалізацію з такою ж сигнатурою, як і метод Where(), тобто можлива заміна Where() на First().
 - Проте слід обережно обирати місце заміни: пряма заміна не дозволить застосовувати сортування:

```
var top =  
    cars.First(c => c.Manufacturer == "BMW" && c.Year == 2016)  
        .OrderByDescending(c => c.Combined)  
        .ThenBy(c => c.Name)  
        .Select(c => c)  
        .First();
```

Фільтрування даних за допомогою Where() та First()

- Спробуємо спочатку все відсортувати:

```
var top =  
    cars  
        .OrderByDescending(c => c.Combined)  
        .ThenBy(c => c.Name)|  
        .Select(c => c)  
        .First(c => c.Manufacturer == "BMW" && c.Year == 2016);
```

- Результат буде правильним, проте ефективність під питанням: раніше сортувався менший набір даних (результат вибірки за критерієм), а зараз – повний набір даних.
 - Оператор First() також відсутній у синтаксисі запитів.
- Разом з First() присутні оператори FirstOrDefault(), Last() та LastOrDefault().

- Якщо не буде жодного елемента даних, який відповідає критеріям вибірки, згенерується виняток

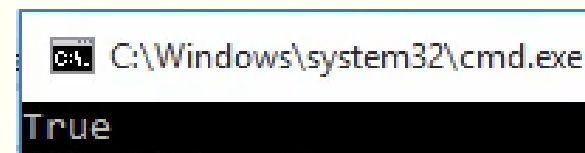
```
Unhandled Exception: System.InvalidOperationException: Sequence contains no matching element  
at System.Linq.Enumerable.First[TSource](IEnumerable`1 source, Func`2 predicate)  
at Cars.Program.Main(String[] args) in C:\dev\linqsamples\Cars\Program.cs:line 22
```

- Інакше у випадку застосування FirstOrDefault() чи LastOrDefault() буде присвоюватись значення за умовчанням (проте це може бути null, що призведе до NullReferenceException). Принаймні, власне запит не генеруватиме виняток.

Квантифікація даних

- Квантифікатори даних вказують, чи щось відповідає критерію-предикату.
 - Квантифікатори повертають булеві значення, тому не пропонують відкладеного виконання.
- Нехай потрібно перевірити, чи є в даних виробник Ford.
 - Використовується квантифікаційний оператор Any().

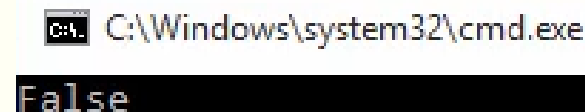
```
var result = cars.Any(c => c.Manufacturer == "Ford");  
  
Console.WriteLine(result);
```



```
C:\Windows\system32\cmd.exe  
True
```

- Чи всі виробники в даних є Ford-ами? Оператор All()

```
var result = cars.All(c => c.Manufacturer == "Ford");
```



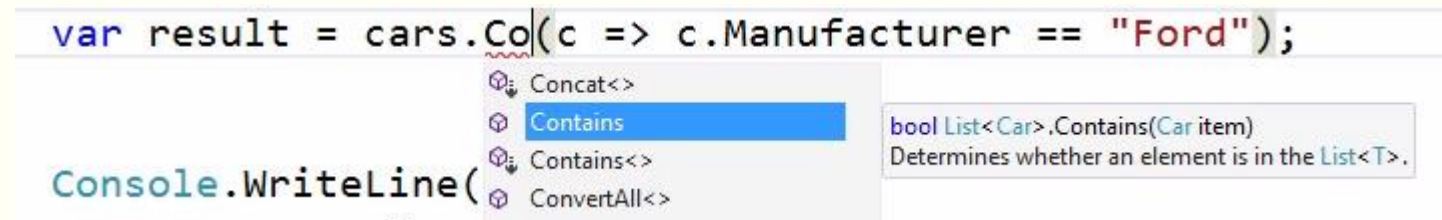
```
C:\Windows\system32\cmd.exe  
False
```

- Хоч All() не передбачає відкладеного виконання, він наближається до лінійної поведінки: як тільки знайдено першу невідповідність критерію, він припиняє подальшу перевірку.

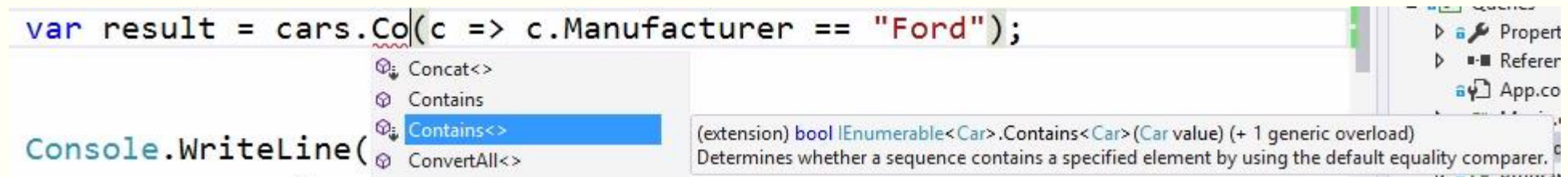
Квантифікація даних

- Також існує кілька версій методу Contains():

- (1) Метод Contains(), визначений для списків:



- (2) Метод розширення Contains<T>() з LINQ:



Проектування даних за допомогою Select()

- Select – основний оператор проектування даних в LINQ.

```
private static List<Car> ProcessFile(string path)
{
    var query =

        File.ReadAllLines(path)
            .Skip(1)
            .Where(l => l.Length > 1)
            .Select(l => Car.ParseFromCsv(l));

    return query;
}
```

(extension) IEnumerable<Car> IEnumerable<string>.Select<string, Car>(Func<string, Car> selector) (+ 1 overload)
Projects each element of a sequence into a new form.

Exceptions:
ArgumentNullException

- Можна також записати запит дещо чистіше: прибрати лямбду в Select(): `.Select(Car.ParseFromCsv);`

Проектування даних за допомогою Select()

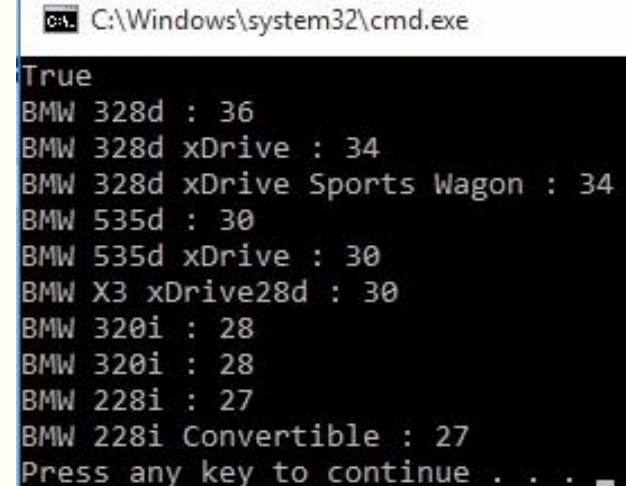
- Ще більше почистити запит дозволяє створення окремого методу розширення ToCar(), що буде поводитись схоже до Select():

```
public static class CarExtensions
{
    public static IEnumerable<Car> ToCar(this IEnumerable<string> source)
    {
        foreach (var line in source)
        {
            var columns = line.Split(',');

            yield return new Car
            {
                Year = int.Parse(columns[0]),
                Manufacturer = columns[1],
                Name = columns[2],
                Displacement = double.Parse(columns[3]),
                Cylinders = int.Parse(columns[4]),
                City = int.Parse(columns[5]),
                Highway = int.Parse(columns[6]),
                Combined = int.Parse(columns[7])
            };
        }
    }
}
```

```
private static List<Car> ProcessFile(string path)
{
    var query =
        File.ReadAllLines(path)
            .Skip(1)
            .Where(l => l.Length > 1)
            .ToCar();

    return query.ToList();
}
```



```
C:\Windows\system32\cmd.exe
True
BMW 328d : 36
BMW 328d xDrive : 34
BMW 328d xDrive Sports Wagon : 34
BMW 535d : 30
BMW 535d xDrive : 30
BMW X3 xDrive28d : 30
BMW 320i : 28
BMW 320i : 28
BMW 228i : 27
BMW 228i Convertible : 27
Press any key to continue . . .
```

Проектування даних за допомогою Select()

- Операції проектування особливо корисні, коли потрібно перетворити послідовність одного типу в послідовність іншого типу.
 - Також застосовуються для формування підмножини даних.
 - Зокрема, тут ми відбираємо лише потрібні стовпці: Manufacturer, Name, Combined з потенційних десятків стовпців з даними.
 - Для цього створюватимемо анонімно типізований об'єкт

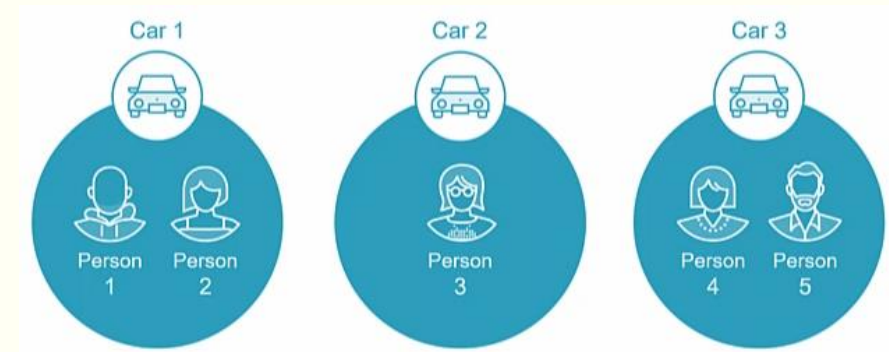
```
var query =  
    from car in cars  
    where car.Manufacturer == "BMW" && car.Year == 2016  
    orderby car.Combined descending, car.Name ascending  
    select new  
    {  
        car.Manufacturer,  
        car.Name,  
        car.Combined  
    };
```

```
var result = cars.Select(c => new { c.Manufacturer,  
                                    c.Name, c.Combined });
```

```
foreach (var car in query.Take(10))  
{  
    Console.WriteLine($"{car.Manufacturer} {car.Name} : {car.Combined}");  
}
```

Сплюснення (flattening) даних за допомогою оператора SelectMany()

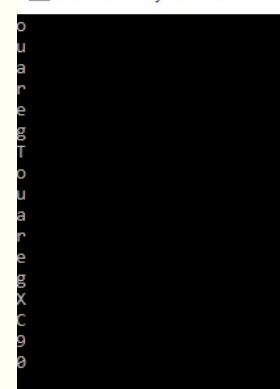
- SelectMany() дозволяє перетворити колекцію колекцій в єдину колекцію.
 - Послідовність послідовностей в єдину послідовність.
 - Наприклад, всередині автомобілів є пасажирів:



- Бажано працювати з даними пасажирів в формі одної послідовності, а не послідовності автомобілів з включеними в них пасажирами.
- Далі в прикладі розглянемо назви автомобілів як рядки – послідовності символів.

```
var result = cars.Select(c => c.Name);  
  
foreach (var name in result)  
{  
    foreach (var character in name)  
    {  
        Console.WriteLine(character);  
    }  
}
```

C:\Windows\system32\cmd.exe



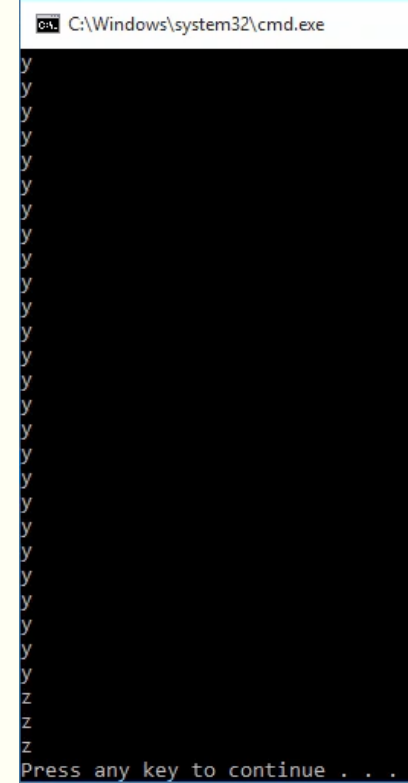
Сплюснення (flattening) даних за допомогою оператора SelectMany()

- Негайний доступ до окремих символів:

```
var result = cars.SelectMany(c => c.Name);  
  
foreach (var character in result)  
{  
    Console.WriteLine(character);  
}
```

- Сортування за символами (можна оцінити частоту появи символів):

```
var result = cars.SelectMany(c => c.Name)  
                    .OrderBy(c => c);  
  
foreach (var character in result)  
{  
    Console.WriteLine(character);  
}
```





ДЯКУЮ ЗА УВАГУ!

Наступне запитання: З'єднання, групування та агрегування даних