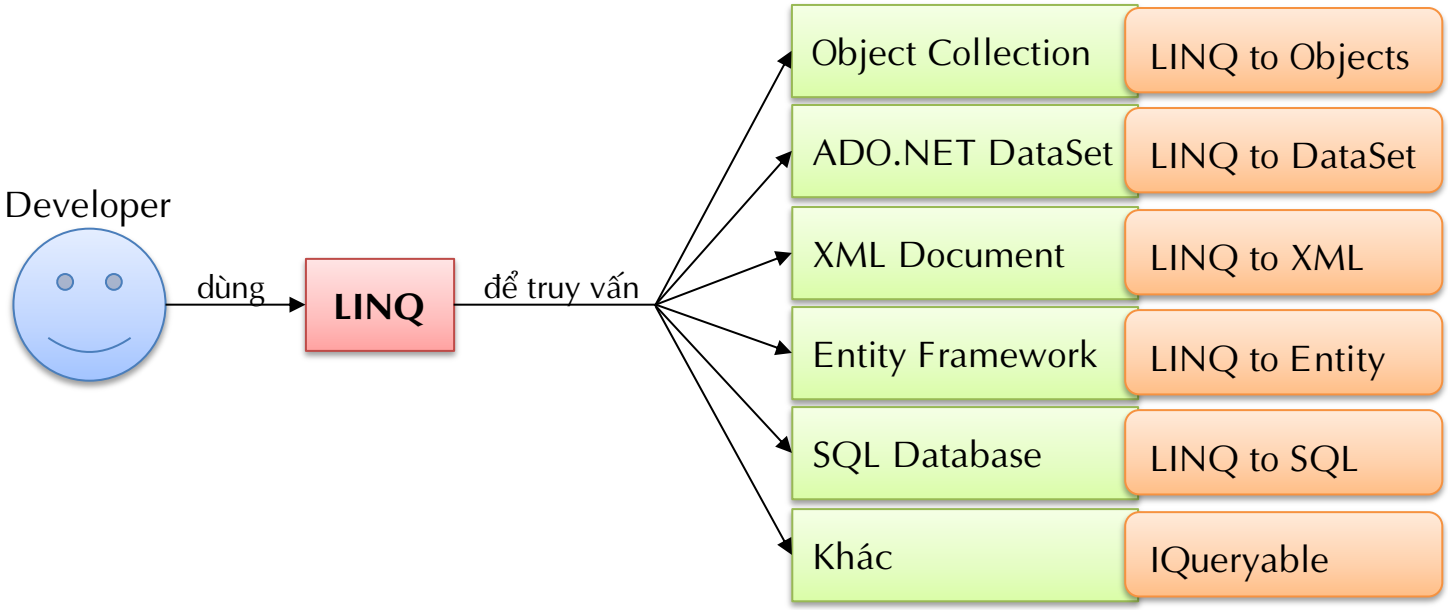
Buổi 7

Truy vấn dữ liệu với LINQ

I. Khái niệm LINQ

Với 1 ứng dụng, dữ liệu có thể được lưu ở nhiều dạng khác nhau. Có thể kể đến như mảng (array), danh sách (list), file XML, file JSON, cơ sở dữ liệu (database)... Với mỗi dạng kể trên, cách duyệt phần tử và truy vấn dữ liệu cũng khác nhau, khiến cho code để truy vấn dữ liệu ở dạng này không thể dùng lại được với dữ liệu ở dạng khác.

LINQ (**L**anguage **In**tegrated **Q**uery) là 1 cú pháp truy vấn được tích hợp trong C#, cho phép truy vấn dữ liệu từ nhiều nguồn, nhiều định dạng khác nhau như Collection, DataSet file XML, file JSON, CSDL... với cú pháp thống nhất:



Các câu truy vấn LINQ trả về kết quả dưới dạng object (hoặc 1 tập hợp object), do đó lập trình viên có thể thao tác trực tiếp với các kết quả này theo các phương pháp hướng đối tượng mà không cần phải chuyển đổi kiểu dữ liệu:

Ví dụ: Đếm số lượng số dương trong 1 danh sách số nguyên:

```
List<int> list = new List<int> { 3, -2, 9, 4, -6, 2, 0, 7 };
Console.WriteLine(list.Count(x => x > 0));    // Xuất ra: 5
```

Trong đó: `Count()` là 1 phương thức của LINQ cho phép đếm số lượng phần tử thỏa mãn điều kiện. Phương thức này có thể được gọi trên bất kỳ Collection nào (trong ví dụ là `List`).

II. Cú pháp

Để có thể sử dụng được LINQ, cần thêm 2 namespace sau (mặc định các file mã nguồn C# đã có sẵn 2 namespace này):

```
using System.Collections.Generic
using System.Linq
```

LINQ có 2 loại cú pháp khác nhau: LINQ Query và LINQ Method

1. LINQ Query

Cú pháp LINQ dưới dạng truy vấn (**LINQ Query**) khá giống với ngôn ngữ SQL, nên những lập trình viên đã quen thuộc với SQL sẽ không mất quá nhiều thời gian để làm quen.

LINQ Query thường bắt đầu bằng mệnh đề `from`, theo sau đó có thể là các mệnh đề như `where`, `orderby`… hoặc các toán tử, cuối cùng kết thúc bằng mệnh đề `select` hoặc `group`:

```
from tên_biến in tên_collection
[các mệnh đề khác]
select/group kết_quả
```

Trong đó:

- `tên_biến` là tên biến do lập trình viên tự đặt, dùng để duyệt từng phần tử trong collection. Có thể xem như mệnh đề `from` này tương tự vòng lặp sau:
  ```
  foreach (var tên_biến in tên_collection)
  ```
- `kết_quả` là biểu thức kết quả.

Ví dụ: Câu truy vấn LINQ lọc ra các số dương trong 1 danh sách số nguyên:

```
var result = from item in list
             where item > 0
             select item;
```

Nhược điểm của LINQ Query là không hỗ trợ hết tất cả các toán tử truy vấn, và cú pháp cũng khó làm quen với các lập trình viên .NET. Ngoài ra, tại thời điểm biên dịch, LINQ Query cũng sẽ được biên dịch thành LINQ Method, do đó, khi xây dựng ứng dụng .NET, khuyến khích sử dụng LINQ Method.

2. LINQ Method

Cú pháp LINQ dưới dạng phương thức (**LINQ Method**) dùng các phương thức để truy vấn. Các phương thức này đều là các phương thức mở rộng cho các kiểu dữ liệu collection.

Ví dụ: Phương thức LINQ lọc ra các số dương trong 1 danh sách số nguyên:

```
var result = list.Where(item => item > 0);
```

Ưu điểm của LINQ Method là hỗ trợ tất cả các toán tử truy vấn, và cú pháp theo kiểu hướng đối tượng sẽ quen thuộc với lập trình viên .NET. Ngoài ra, cách phương thức LINQ còn có thể được gọi nối tiếp nhau, giúp thực hiện được các yêu cầu truy vấn phức tạp.

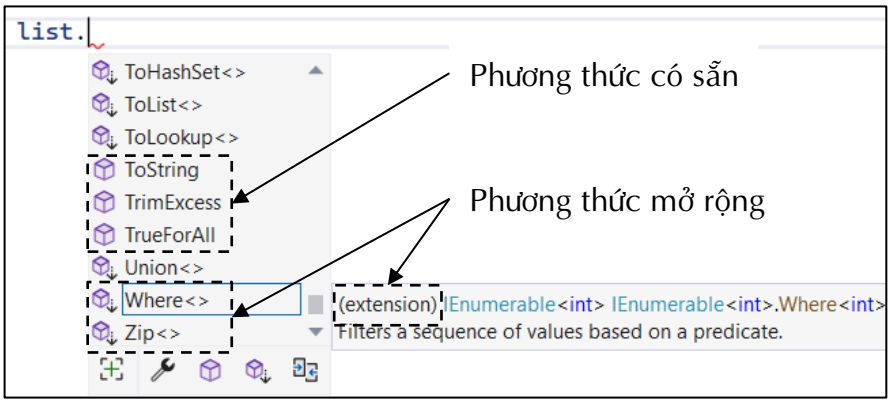
III. Extension Method

1. Khái niệm

Các phương thức LINQ như `Count()` hay `Where()` trong các ví dụ trên đều không có sẵn trong class `List`. Do đó, để 1 danh sách có thể gọi được các phương thức này, thì chúng phải được cài đặt bổ sung vào class `List`.

Các phương thức được cài đặt bổ sung vào 1 class như vậy được gọi là phương thức mở rộng (**extension method**), cho phép lập trình viên bổ sung thêm chức năng cho 1 class nào đó mà không cần phải chỉnh sửa hay kế thừa lại class đó. Các phương thức này có thể được cài đặt cho class tự tạo, class của .NET Framework, hay các class của bên thứ ba.

Trong Visual Studio, extension method có biểu tượng và ghi chú khác so với các phương thức có sẵn của class để giúp lập trình viên dễ nhận biết (xem hình bên):



2. Cách cài đặt

Do class `List` không có phương thức để xuất danh sách ra màn hình, nên mỗi khi muốn xuất 1 danh sách kiểu `List<int>`, chúng ta luôn phải viết vòng lặp để duyệt toàn bộ danh sách:

```
foreach (var item in list)
{
    Console.WriteLine($"{item} ");
}
```

Việc này khá bất tiện và mất thời gian. Do đó, chúng ta sẽ cài đặt extension method cho kiểu `List<int>` để xuất danh sách ra màn hình.

Quy tắc cài đặt 1 extension method:

- Access modifier phải là `public`.
- Phương thức phải là `static` và đặt trong 1 class cũng là `static`.
- Tham số đầu tiên là kiểu mà phương thức này đang cài đặt bổ sung (trong ví dụ này là `List<int>`) và từ khóa `this`. Sau đó là các tham số còn lại của phương thức.

Phương thức `Output()` mở rộng cho `List<int>` như sau:

```
public static class ExtensionMethod
{
      public static void Output(this List<int> list)
      {
            foreach (var item in list)
            {
                  Console.WriteLine($"{item} ");
            }
      }
}
```

Sau khi đã cài đặt, extension method có thể được gọi như 1 phương thức bình thường:

```
List<int> list = new List<int> { 3, -2, 9, 4, -6, 2, 0, 7 };

// Xuất toàn bộ danh sách ra màn hình
list.Output();

// Xuất các số dương trong danh sách ra màn hình
list.Where(x => x > 0).ToList().Output();
```

Extension method

Trong đó: Phương thức `ToList()` dùng để chuyển đổi kiểu dữ liệu về `List` (sẽ được trình bày ở phần IV.7).

3. Extension method trong LINQ

Như đã phân tích ở phần III.1, các phương thức truy vấn của LINQ đều là extension method trên các kiểu dữ liệu Collection.

Lấy ví dụ phương thức `Count()` dùng để đếm số phần tử thỏa mãn điều kiện được .NET Framework khai báo và cài đặt như sau:

```
public static int Count<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate)
{
    if (source == null)
    {
        throw Error.ArgumentNull("source");
    }

    if (predicate == null)
    {
        throw Error.ArgumentNull("predicate");
    }

    int num = 0;
    foreach (TSource item in source)
    {
        if (predicate(item))
        {
            num = checked(num + 1);
        }
    }

    return num;
}
```

Xét các trường hợp đặc biệt

Đoạn code xử lý chính

Phần khai báo phương thức:

```
public static int Count<TSource>(this IEnumerable<TSource> source,
Func<TSource, bool> predicate) { … }
```

Trong đó:

- `IEnumerable` là 1 interface được class `List` triển khai.
- `TSource` đại diện cho 1 kiểu dữ liệu bất kỳ. Trong ví dụ này thì `TSource` là `int`.

Để cho dễ hiểu, chúng ta có thể viết lại phần khai báo phương thức `Count()` như sau:

```
public static int Count<int>(this List<int> source, Func<int, bool>
predicate) { … }
```

Xét về mặt cú pháp, phương thức `Count()` là 1 extension method cho kiểu `IEnumerable` (và do đó `List` cũng có thể gọi phương thức này). Nếu bỏ qua tham số đầu tiên là `List<int>`, thì phương thức này nhận 1 tham số `predicate` là 1 delegate có dạng `(int) => bool`.

Dễ thấy rằng đoạn code xử lý chính của phương thức `Count()` chính là đoạn code rất quen thuộc dùng để đếm số lượng phần tử trong 1 tập hợp thỏa mãn 1 điều kiện nào đó:

```
int num = 0;
foreach (int item in source)
{
      if (predicate(item))
      {
            num = checked(num + 1);
      }
}
return num;
```

Điểm khác biệt duy nhất đó là giờ đây, điều kiện không được ghi trực tiếp trong câu lệnh `if` nữa mà sẽ được gọi thông qua 1 delegate. Giả sử người dùng cần đếm số lượng số dương trong danh sách thì sẽ gọi phương thức Count() như sau:

```
list.Count(x => x > 0);
```

Khi đó, tham số `predicate` sẽ có giá trị là `x => x > 0`, hay nói cách khác, tham số `predicate` trỏ tới 1 phương thức vô danh `x => x > 0`. Do đó, khi gọi `predicate` sẽ chính là gọi phương thức vô danh này.

Nhờ có extension method và delegate, các phương thức truy vấn của LINQ rất linh hoạt, có thể nhận vào bất kỳ biểu thức điều kiện nào được viết dưới dạng delegate, anonymous method hoặc biểu thức Lambda.

Ví dụ: Gọi phương thức `Count()` để đếm số lượng số dương và truyền biểu thức điều kiện vào tham số dưới dạng:

- Delegate:

```
list.Count(CheckPositive);

bool CheckPositive(int x) { return x > 0; }
```

- Anonymous method:

```
list.Count(delegate (int x) { return x > 0; });
```

- Biểu thức Lambda:

```
list.Count(x => x > 0);
```

Cả 3 cách viết trên đều đúng và cho kết quả như nhau, do đó với LINQ, chúng ta sẽ dùng biểu thức Lambda cho ngắn gọn và dễ đọc.

IV. Các phương thức truy vấn của LINQ

Mặc dù LINQ hỗ trợ 2 cú pháp, tuy nhiên LINQ Method có nhiều ưu điểm hơn và phổ biến hơn, do đó trong môn học này chúng ta sẽ dùng cú pháp LINQ Method.

Có trên 50 phương thức LINQ được chia thành nhiều nhóm khác nhau dựa vào mục đích, ví dụ như lọc dữ liệu (filtering), sắp xếp (sorting), tập hợp (aggregation)... Trong bảng dưới đây, những nhóm phương thức được tô màu vàng là những nhóm thường dùng nhất:

Nhóm	Phương thức
Filtering	`Where`
Sorting	`OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse`
Grouping	`GroupBy, ToLookUp`
Projection	`Select`
Join	`GroupJoin, Join`
Aggregation	`Aggregate, Average, Count, LongCount, Max, Min, Sum`
Quantifiers	`All, Any, Contains`
Element	`ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault`
Set	`Distinct, Except, Intersect, Union`
Partitioning	`Skip, SkipWhile, Take, TakeWhile`
Relation	`Include`
Conversion	`AsEnumerable, AsQueryable, ToArray, ToDictionary, ToList, Cast`

Các phương thức LINQ đều là phương thức mở rộng của `IEnumerable` và `IQueryable`. Đa số các phương thức LINQ đều nhận vào tham số là 1 biểu thức Lambda và trả về kết

quả dưới dạng `IEnumerable` hoặc tương đương, do đó chúng ta có thể gọi liên tiếp nhiều phương thức LINQ trong cùng 1 câu lệnh.

Ngoài ra, các phương thức LINQ khác nhau có thể trả về kiểu dữ liệu khác nhau. Do đó, nếu muốn gán kết quả của các phương thức này cho 1 biến, nên khai báo biến đó bằng `var`:

```
var result = list.Where(x => x > 0);
```

Trong các ví dụ sau đây, chúng ta sẽ dùng LINQ Method để truy vấn 2 danh sách:

- Danh sách số nguyên `list`:

```
List<int> list = new List<int> { 3, -2, 9, 4, -6, 2, 0, 7 };
```

- Danh sách sinh viên `students`:

```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public double GPA { get; set; }
}
List<Student> students = new List<Student>
{
    new Student { Id = 1, Name = "Andy", Age = 18, GPA = 5.9 },
    new Student { Id = 2, Name = "Brent", Age = 19, GPA = 3.5 },
    new Student { Id = 3, Name = "Cody", Age = 18, GPA = 2.7 },
    new Student { Id = 4, Name = "David", Age = 20, GPA = 9.8 },
    new Student { Id = 5, Name = "Eric", Age = 22, GPA = 7.3 },
    new Student { Id = 6, Name = "Frank", Age = 20, GPA = 8.2 }
};
```

1. Lọc dữ liệu theo điều kiện (Filtering)

Phương thức `Where()` cho phép lọc ra các phần tử thỏa mãn điều kiện:

- Lọc ra danh sách số dương:

```
list.Where(x => x > 0);
```

- Lọc ra danh sách số dương chẵn:

```
list.Where(x => x > 0).Where(x => x % 2 == 0);
```

- Lọc ra danh sách sinh viên ≥ 20 tuổi:

```
students.Where(s => s.Age >= 20);
```

- Lọc ra danh sách sinh viên ≥ 20 tuổi và có điểm trung bình ≥ 5:

```
students.Where(s => s.Age >= 20).Where(s => s.GPA >= 5);
```

Mặc dù phương thức `Where()` có thể gọi liên tiếp nhau, nhưng như vậy sẽ khiến chương trình phải thực hiện truy vấn 2 lần, dẫn đến tốn thời gian và tài nguyên của máy. Do đó, chúng ta nên gộp nhiều phương thức `Where()` liên tiếp lại làm 1 như sau:

- Lọc ra danh sách số dương chẵn:

```
list.Where(x => x > 0 && x % 2 == 0);
```

- Lọc ra danh sách sinh viên ≥ 20 tuổi và có điểm trung bình ≥ 5:

```
students.Where(s => s.Age >= 20 && s.GPA >= 5);
```

2. Sắp xếp (Sorting)

Phương thức `OrderBy()` cho phép sắp xếp collection theo 1 thuộc tính. Sắp xếp theo thuộc tính thứ 2 trở đi phải dùng phương thức `ThenBy()`. Tuy nhiên, 2 phương thức này chỉ sắp xếp tăng dần. Để sắp xếp giảm dần, cần dùng 2 phương thức tương ứng là `OrderByDescending()` và `ThenByDescending()`:

- Sắp xếp danh sách số nguyên giảm dần:

```
list.OrderByDescending(x => x);
```

- Sắp xếp danh sách sinh viên theo số tuổi tăng dần:

```
students.OrderBy(s => s.Age);
```

- Sắp xếp danh sách sinh viên theo số tuổi tăng dần. Nếu tuổi bằng nhau thì sắp xếp theo điểm trung bình giảm dần:

```
students.OrderBy(s => s.Age).ThenByDescending(s => s.GPA);
```

3. Tính toán tổng hợp (Aggregation)

Phương thức `Sum()`, `Max()`, `Min()`, `Average()` cho phép tính tổng, tìm giá trị lớn nhất, nhỏ nhất, tính trung bình cộng của các giá trị số. Phương thức `Count()` cho phép đếm số lượng phần tử thỏa mãn điều kiện:

- Tính tổng của danh sách số nguyên:

```
list.Sum();
// HOẶC
list.Sum(x => x);
```

- Cho biết giá trị tuổi cao nhất trong danh sách sinh viên:

```
students.Max(s => s.Age);
```

- Đếm số lượng sinh viên đậu (có điểm trung bình ≥ 5):

```
students.Count(s => s.GPA >= 5);
```

Lưu ý:

- Trong ví dụ trên, phương thức `Max()` chỉ cho biết giá trị tuổi cao nhất. Muốn lọc ra danh sách sinh viên lớn tuổi nhất, cần phải truy vấn như sau:

```
int maxAge = students.Max(s => s.Age);
students.Where(s => s.Age == maxAge);
```

Cần phân biệt property `Count` và phương thức `Count()` của collection. Property `Count` là thành phần có sẵn trong collection, cho biết số lượng phần tử của nó; còn phương thức `Count()` là phương thức mở rộng do LINQ cài đặt vào collection.

4. Định lượng (Quantifier)

Phương thức `All()` trả về `true` nếu tất cả phần tử trong collection thỏa mãn điều kiện. Phương thức `Any()` trả về `true` nếu ít nhất 1 phần tử trong collection thỏa mãn điều kiện:

- Cho biết danh sách số nguyên có phải gồm toàn số dương hay không:

```
list.All(x => x > 0);                    // Trả về: false
```

- Cho biết có sinh viên nào phải thi lại (điểm trung bình < 5) hay không:

```
students.Any(s => s.GPA < 5);            // Trả về: true
```

Phương thức `Contains()` trả về `true` nếu tồn tại phần tử trong collection mang 1 giá trị nào đó, và trả về `false` trong trường hợp ngược lại, ví dụ: Cho biết trong danh sách số nguyên có chứa số 9 hay không:

```
list.Contains(9);    // Trả về: true
```

Tuy nhiên, phương thức `Contains()` không hoạt động đúng với các dữ liệu thuộc dạng tham chiếu như class. Trong ví dụ sau đây, phương thức `Contains()` trả về `false` mặc dù nhận tham số là 1 sinh viên có thông tin giống hệt phần tử `students[0]`:

```
var s = new Student { Name = "Andy", Age = 18, GPA = 5.9 };
students.Contains(students[0]);     // Trả về: true
students.Contains(s);               // Trả về: false
```

Lý do cho việc này là chương trình không biết cách so sánh 2 đối tượng `Student`, nên mặc dù có thông tin giống nhau, nhưng `students[0]` và `s` vẫn là 2 đối tượng khác nhau. Để khắc phục việc này, trong class `Student` cần override phương thức `Equals()`[1] để cài đặt cách so sánh 2 đối tượng kiểu `Student`:

```
public class Student
{
    public override bool Equals(object obj)
    {
        var student = (Student)obj;
        return Name == student.Name &&
               Age == student.Age &&
               GPA == student.GPA;
    }
}
```

→ 2 đối tượng Student bằng nhau khi có thông tin giống hệt nhau

[1] Phương thức `Equals()` được khai báo ở class `object`

5. Chọn phần tử (Element)

a) Phương thức `First()` và `Last()`

Nếu không nhận vào biểu thức Lambda làm tham số, các phương thức `First()` và `Last()` lần lượt lấy ra phần tử đầu tiên và phần tử cuối cùng trong collection:

- Lấy ra số nguyên đầu tiên trong danh sách:

```
list.First();
```

- Lấy ra sinh viên cuối cùng trong danh sách:

```
students.Last();
```

Nếu nhận vào biểu thức Lambda làm tham số, các phương thức `First()` và `Last()` lần lượt lấy ra phần tử đầu tiên và phần tử cuối cùng trong collection thỏa mãn điều kiện quy định bởi biểu thức Lambda này:

- Lấy ra số chẵn đầu tiên trong danh sách:

```
list.First(x => x % 2 == 0);
```

- Lấy ra sinh viên cuối cùng có tuổi ≤ 20 trong danh sách:

```
students.Last(s => s.Age <= 20);
```

Với yêu cầu tìm phần tử đầu tiên và phần tử cuối cùng thỏa mãn điều kiện, có thể dùng phương thức `Where()` kết hợp với `First()`/`Last()`. Ví dụ để tìm sinh viên đầu tiên có điểm trung bình ≥ 5, chúng ta có 2 cách như sau:

```
// Cách 1: Dùng Where() kết hợp với First()
students.Where(s => s.GPA >= 5).First();

// Cách 2: Chỉ dùng First()
students.First(s => s.GPA >= 5);
```

Cách 1 sẽ duyệt toàn bộ danh sách để lọc ra những sinh viên thỏa mãn điều kiện, sau đó từ danh sách kết quả này mới lấy ra phần tử đầu tiên. Trong khi đó, cách 2 sẽ duyệt toàn bộ danh sách, nhưng ngay khi gặp sinh viên đầu tiên thỏa mãn điều kiện sẽ trả về kết quả. Do đó, cách 2 thực thi nhanh hơn và ít tốn bộ nhớ hơn so với cách 1.

b) Phương thức `ElementAt()` và `Single()`

Phương thức `ElementAt()` trả về phần tử tại 1 vị trí nào đó trong collection. Phương thức `Single()` trả về phần tử duy nhất trong collection (có thể kèm theo điều kiện), hoặc báo lỗi nếu danh sách không chứa duy nhất 1 phần tử:

- Lấy ra sinh viên ở vị trí thứ 3 trong danh sách:

```
students.ElementAt(3);
// HOẶC
students[3];
```

– Lấy ra số chia hết cho 7 duy nhất trong danh sách:

```
list.Single(x => x % 7 == 0);        // Trả về: 7
```

– Lấy ra sinh viên duy nhất trong lớp:

```
students.Single();        // InvalidOperationException
```

c) Các phương thức `OrDefault`:

Các phương thức chọn phần tử đều sẽ báo lỗi khi không tìm ra được phần tử phù hợp:

Phương thức	Loại lỗi	Trường hợp báo lỗi
`First()`; `Last()`	`InvalidOperation`	Collection không có phần tử thỏa mãn điều kiện
`ElementAt()`	`ArgumentOutOfRange`	Vị trí phần tử nằm ngoài collection
`Single()`	`InvalidOperation`	Collection không chứa duy nhất 1 phần tử thỏa mãn điều kiện

Nếu không muốn gặp những lỗi như vậy, chúng ta có thể dùng các biến thể của các phương thức trên, lần lượt là `FirstOrDefault()`, `LastOrDefault()`, `ElementAtOrDefault()` và `SingleOrDefault()`. Thay vì báo lỗi, các phương thức này sẽ trả về giá trị mặc định của kiểu dữ liệu đang được truy vấn. Trong ví dụ: giá trị mặc định của kiểu `int` là `0` và của kiểu `Student` (hay các kiểu class nói chung) là `null`.

Riêng phương thức `SingleOrDefault()` sẽ chỉ trả về giá trị mặc định khi collection không chứa phần tử thỏa mãn điều kiện, còn nếu có nhiều hơn 1 phần tử thỏa mãn điều kiện thì vẫn sẽ báo lỗi `InvalidOperation`.

Lưu ý: Không phải lúc nào chúng ta cũng dùng các phương thức `OrDefault` thay cho các phương thức gốc, vì các lỗi có dạng Exception[2] kể trên đều có thể xử lý được.

6. Phân vùng (Partitioning)

Phương thức `Skip()` cho phép bỏ qua một lượng phần tử cụ thể ở đầu danh sách. Phương thức `Take()` cho phép lấy một lượng phần tử cụ thể ở đầu collection. Kết hợp cả `Skip()` và `Take()` cho phép lấy 1 đoạn phần tử cụ thể trong collection:

– Lấy 4 số nguyên ở đầu danh sách:

```
list.Take(4);
```

– Lấy danh sách sinh viên, kể từ sinh viên thứ 3:

```
students.Skip(2);
```

[2] Sẽ được trình bày ở buổi 16

– Lấy danh sách sinh viên thứ 3 đến thứ 5:

```
students.Skip(2).Take(3);
```

7. Chuyển đổi kiểu dữ liệu (Conversion)

Một số phương thức LINQ như `Where()`, `Skip()`, `Take()`, `OrderBy()`… trả về kết quả dưới dạng `IEnumerable`. Chúng ta có thể chuyển kết quả này thành các kiểu collection quen thuộc hơn như `Array` (mảng), `List` (danh sách) và `Dictionary`:

– Phương thức `ToArray()` chuyển đổi thành `Array`:

```
var result = students.ToArray();
```

Kiểu `Student[]`

– Phương thức `ToList()` chuyển đổi thành `List`:

```
var result = students.Where(s => s.Age >= 20).ToList();
```

Kiểu `IEnumerable<Student>`

Kiểu `List<Student>`

– Phương thức `ToDictionary()` chuyển đổi thành `Dictionary` có key là `Id` của Sinh viên, và value là toàn bộ object Sinh viên đó:

```
var result = students.ToDictionary(s => s.Id);
```

Kiểu `Dictionary<int, Student>`

Ngoài ra, các phương thức ở trên cũng hỗ trợ ghi tường minh kiểu dữ liệu cho collection:

```
students.ToArray<Student>();
students.Where(s => s.Age >= 20).ToList<Student>();
```

V. Deferred Execution và Immediate Execution

Các câu truy vấn trong LINQ có thể được thực thi tức thời (**immediate execution**) hoặc thực thi trì hoãn (**deferred execution**).

1. Thực thi trì hoãn (Deferred Execution)

Xét đoạn lệnh sau tìm các số âm trong danh sách và xuất ra màn hình:

```
var result = list.Where(x => x < 0);
list.Add(-10);
foreach (var item in result)
{
    Console.Write($"{item} ");
}
```

Mặc dù trong danh sách chỉ có 2 số âm là -2 và -6, còn số -10 được thêm vào sau khi câu truy vấn được tạo ra, nhưng kết quả xuất ra màn hình vẫn có đầy đủ 3 số này. Điều này được giải thích là các phương thức truy vấn LINQ không được thực thi vào thời điểm chúng được tạo ra, mà sẽ được thực thi tại thời điểm cần đến kết quả truy vấn (trong ví dụ trên là

khi duyệt kết quả bằng vòng lặp `foreach`). Việc này được gọi là thực thi trì hoãn (**deferred execution**).

Deferred execution cho lập trình viên sự linh hoạt trong việc viết truy vấn, có thể tách 1 câu truy vấn phức tạp thành nhiều câu truy vấn con. Ngoài ra, thực thi theo cách này giúp dữ liệu lấy được luôn là mới nhất, rất hữu ích trong trường hợp dữ liệu thay đổi liên tục.

2. Thực thi tức thời (Immediate Execution)

Nhược điểm của thực thi trì hoãn là nếu dùng với dữ liệu không có nhiều thay đổi thì cứ mỗi lần cần lấy dữ liệu, câu truy vấn lại phải thực thi 1 lần. Điều này sẽ tiêu tốn tài nguyên và giảm hiệu suất của chương trình. Thay vào đó, ta có thể ép câu truy vấn phải thực thi ngay lúc nó được tạo ra. Cách này được gọi là thực thi tức thời (**immediate execution**).

Xét ví dụ trên, nhưng câu lệnh truy vấn thay đổi như sau:

```
var result = list.Where(x => x < 0).ToList();
list.Add(-10);
foreach (var item in result)
{
      Console.Write($"{item} ");
}
```

Các phương thức chuyển đổi kiểu dữ liệu[3] ép câu truy vấn phải thực thi ngay lập tức.

VI. Bài tập

Thực hiện các bài tập sau, mỗi bài nằm trong 1 project của solution **OOP_Buoi12**.

Bài 1: Cho danh sách sản phẩm của 1 cửa hàng bán sách như sau:

Id	SKU	Name	Author	Price	Stock
1	WT3WPG	Tuổi trẻ đáng giá bao nhiêu	Rosie Nguyễn	45000	40
2	98IOWW	Luyện thi THPT Quốc gia – Hóa học	Nguyễn Đức Dũng	51000	15
3	21RH48	Khéo ăn khéo nói sẽ có được thiên hạ	Trác Nhã	59000	29
4	QOXYSD	Nhà giả kim	Paulo Coelho	53000	3
5	6YI6TZ	Để yên cho bác sĩ "Hiền"	BS. Ngô Đức Hùng	45000	36
6	YHB5JT	Mình là cá, việc của mình là bơi	Takeshi Furukawa	51000	9
7	LXL64L	Đời ngắn đừng ngủ dài	Robin Sharma	42000	7
8	C5V645	Luyện thi THPT Quốc gia – Toán	ThS. Đỗ Đường Hiếu	51000	0
9	4KLYT2	Cà phê cùng Tony	Tony Buổi Sáng	62000	18

[3] Xem phần IV.7

10	KBD67V	Em sẽ đến cùng cơn mưa	Ichikawa Takuji	56000	64
11	3RISEF	Quẳng gánh lo đi mà vui sống	Dale Carnegie	45000	120
12	VIAZXR	Mình nói gì khi nói về hạnh phúc?	Rosie Nguyễn	36000	0

Yêu cầu:

- Xây dựng các class để lưu trữ thông tin 1 cuốn sách và lưu trữ danh sách các cuốn sách trong cửa hàng.
- Liệt kê danh sách tất cả các cuốn sách trong cửa hàng, sắp xếp theo giá tiền tăng dần. Nếu giá tiền bằng nhau thì sắp xếp theo số lượng tồn kho giảm dần.
- Liệt kê danh sách các cuốn sách gần hết hàng (có số lượng tồn kho < 10).
- Liệt kê danh sách các cuốn sách đã hết hàng.
- Liệt kê danh sách các cuốn sách của tác giả Rosie Nguyễn và còn hàng.
- Liệt kê top 5 cuốn sách có giá cao nhất.
- Liệt kê tất cả cuốn sách có giá cao hơn 50000.
- Tính giá tiền trung bình của 1 cuốn sách.
- Tính tổng tiền hàng còn tồn kho.