**LINQ**

LINQ (Language Integrated Query) kavramı C# içerisinde sorgulama işlemlerinin kolay yapılması için dile eklenmiş bir teknolojidir. LINQ genel olarak bir nesneye, veritabanına ya da bir XML dosyasına ilişkin olabilmektedir. İngilizce olarak örneğin veritabanına ilişkin LINQ yapısına “LINQ to Sql” denilmektedir. Linq istenirse herhangi bir yapıya ilişkin de tasarlanabilir. C# içerisinde LINQ iki şekilde kullanılabilir.

1. Linq sorgu sentaksı (Linq query syntax)

2. Linq metot sentaksı (Linq method syntax)

İki yöntemin de kullanımı duruma göre okunabiliği/algılanabilirliği artırabilmektedir.

**Linq Sorgu Sentaksı**

Linq’in klasik SQL sentaksına benzeyen bir sentaks yapısı vardır. Bu sentaks yapısı C# içerisinde kullanılabilir. Linq sorgu sentaksınının sonucu sorguya ilişkin bilgidir. Bu bilgi genel olarak bir collection referansıdır. Bu bilginin ne türden bir collection ile verileceğinin bir standardı yoktur. Pratikte bu türün ne olduğunun programcı açısından da bir önemi yoktur. Bu sebeple sorgunun sonucu ***var*** bir değişkene atanmalıdır. Linq sorgu sentaksı ile bir dizi içerisindeki sıfırdan büyük ve çift olan sayıların elde edildiği program aşağıdaki gibi yazılabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, -2, 3, 4, 5, 6, 7 };

var query = from val in a

where val % 2 == 0 && val > 0

select val;

Console.WriteLine(query.GetType().Name);

foreach (int val in query)

Console.WriteLine(val);

}

}

}

***Anahtar Notlar:*** *Bazen bir referans ile bir sınıfın birden fazla metodunun peşpeşe çağrılması gerekebilir. Bu durumda kolay kullanım açısından metotlar this referansı ile döndürülür ve zincirleme bir çağırma yapılır. Bu yönteme “fluent pattern” denilmektedir. Örneğin:*

***using System;***

*using System.Collections.Generic;*

*using System.Linq;*

*namespace CSD*

*{*

*class Sample {*

*public Sample Foo()*

*{*

*Console.WriteLine("Foo");*

*return this;*

*}*

*public Sample Bar()*

*{*

*Console.WriteLine("Bar");*

*return this;*

*}*

*public Sample Tar()*

*{*

*Console.WriteLine("Tar");*

*return this;*

*}*

*}*

*public class App*

*{*

*public static void Main()*

*{*

*Sample s = new Sample();*

*s.Foo().Bar().Tar();*

*}*

*}*

*}*

***Anahtar Notlar:***  *Yukarıdaki kalıba benzer şekilde bir sınıf türünden referans ile çağırmaya başlanıp, başka bir sınıfın da metodu zincir içerisinde çağrılabilir. Örneğin:*

***using System;***

*using System.Collections.Generic;*

*using System.Linq;*

*namespace CSD*

*{*

*class Mample {*

*public Mample Tar()*

*{*

*Console.WriteLine("Mample.Tar");*

*return this;*

*}*

*}*

*class Sample {*

*public Sample Foo()*

*{*

*Console.WriteLine("Foo");*

*return this;*

*}*

*public Sample Bar()*

*{*

*Console.WriteLine("Bar");*

*return this;*

*}*

*public Mample Tar()*

*{*

*Console.WriteLine("Sample.Tar");*

*return new Mample();*

*}*

*}*

*public class App*

*{*

*public static void Main()*

*{*

*Sample s = new Sample();*

*s.Foo().Bar().Tar().Tar();*

*}*

*}*

*}*

Linq to Object tüm IEnumerable arayüzünü destekleyen sınıflar için çalışabilmektedir. Örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

List<int> list = new List<int> { 1, 2, 3, -4, 5};

var query = from val in list

where val % 2 == 0 && val > 0

select val;

foreach (var val in query)

Console.WriteLine(val);

}

}

}

Collection sınıfların açılımlarına ilişkin türlerin de elemanları sorgu sentaksı içerisinde kullanılabilmektedir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product {

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> list = new List<Product> {

new Product { Id=1, Name = "Muz", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "Elma", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "Armut", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = from p in list

where p.ExpiryDate < DateTime.Today

select p;

foreach (var p in query)

Console.WriteLine(p);

}

}

}

**Linq Metot Sentaksı**

IEnumerable arayüzüne eklenti olarak bildirilmiş metotlar ile zincir çağırması şeklinde yapılan sorgulardır. Bu arayüzün Linq to Sql karşılığı IQueryable arayüzüdür Örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product {

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> list = new List<Product> {

new Product { Id=1, Name = "Muz", ExpiryDate = new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "Elma", ExpiryDate = new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "Armut", ExpiryDate = new DateTime(2016, 3, 21)},

};

var query = list.Where(p => p.ExpiryDate < DateTime.Today);

foreach (var p in query)

Console.WriteLine(p);

}

}

}

Linq içerisinde anonim türler de oluşturulabilmektedir. Hatta bu türlerin her bir property elemanına ayrı isimler de verilebilir.

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product {

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> list = new List<Product> {

new Product { Id=1, Name = "Muz", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "Elma", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "Armut", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = list.Where(p => p.ExpiryDate < DateTime.Today).Select(p => new { ProductName = p.Name, Date = p.ExpiryDate});

foreach (var p in query)

{

Console.WriteLine("{0}-{1}", p.ProductName, p.Date.ToShortDateString());

}

}

}

}

Linq sorgu sentaksı ile de benzer işlem yapılabilmektedir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product {

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> list = new List<Product> {

new Product { Id=1, Name = "Muz", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "Elma", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "Armut", ExpiryDate=new DateTime(2020, 3, 21)},

};

var query = from p in list

where p.ExpiryDate < DateTime.Today

select new { ProductName = p.Name, Date = p.ExpiryDate};

foreach (var p in query)

{

Console.WriteLine("{0}-{1}", p.ProductName, p.Date.ToShortDateString());

}

}

}

}

Linq sorgu sentaksı ile bileşik (kartezyen) sorgulamalar da yapılabilmektedir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

public List<Order> Orders { get; set; } = new List<Order>();

}

class Customer

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime BirthDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, BirthDate.ToShortDateString());

}

public List<Order> Orders { get; set; } = new List<Order>();

}

class Order

{

public Customer Customer { get; set; }

public Product Product { get; set; }

public int Count { get; set; }

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

};

List<Customer> customers = new List<Customer>() {

new Customer { Id = 1, Name = "csd", BirthDate= DateTime.Today },

new Customer { Id = 2, Name = "msd", BirthDate= DateTime.Today },

};

List<Order> orders = new List<Order>() {

new Order { Customer = customers[0], Product = products[1], Count = 3},

new Order { Customer = customers[1], Product = products[0], Count = 2},

new Order { Customer = customers[1], Product = products[2], Count = 5},

};

var query = from p in products

from c in customers

from o in orders

where p.Id == o.Product.Id && c.Id == o.Customer.Id && o.Count >= 3

select new { CustomerName = c.Name, ProductName = p.Name, o.Count };

foreach (var obj in query)

Console.WriteLine(obj);

}

}

}

Linq eklenti metotlarına genel olarak operatör denilmektedir. Tüm operatörler sorgu sentaksında kullanılamamaktadır. Eklenti metotlar IEnumerable ya da IQueryable arayüzlerine eklenti olarak bildirilmiştir. Yani aslında bir operatör dendiğinde genel olarak eklenti metot anlaşılabilir.

**Func Delegeleri**

Hemen hemen tüm eklenti metotlarda Func isimli delegeler kullanılmaktadır.

**Where Metodu**

Koşul belirlemekte kullanılan metottur. Where metodunun iki tane versiyonu vardır:

*public static IEnumerable <T> Where<T>(this IEnumerable <T> ie, Func<T, bool> pred) {/\*...\*/}*

*public static IEnumerable <T> Where<T>(this IEnumerable <T> ie, Func<T, int, bool> pred) {/\*...\*/}*

Bu metot koşula ilişkin bir predicate alır. Örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product {

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2021, 3, 21)},

};

var query = products.Where(p => p.ExpiryDate < DateTime.Today);

foreach (var p in query)

Console.WriteLine(p);

}

}

}

Yukarıdaki programın sorgu sentaksı ile yazılış biçimi:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2021, 3, 21)},

};

var query = from p in products

where p.ExpiryDate < DateTime.Today

select p;

foreach (var p in query)

Console.WriteLine(p);

}

}

}

İkinci versiyonundaki index parametresi ise her elemanının sırasını belirten değerdir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product {

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=1, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=1, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = products.Where((p, index) => p.ExpiryDate < DateTime.Today && index >= 1);

foreach (var p in query)

Console.WriteLine(p);

}

}

}

***Anahtar Notlar:*** *Predicate test amaçlı kullanılan bool türden geri dönüş değeri olan metotlara verilen genel bir isimdir. Where metotlarının implementasyonu aşağıdaki gibi yapılabilir. Aşağıdaki yazımda bazı detaylar gözönünde bulundurulmamıştır:*

***using System;***

*using System.Collections.Generic;*

*using System.Linq;*

*namespace CSD*

*{*

*class Product {*

*public int Id { get; set; }*

*public string Name { get; set; }*

*public DateTime ExpiryDate { get; set; }*

*public override string ToString()*

*{*

*return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());*

*}*

*}*

*public static class MyEnumerable {*

*public static IEnumerable<T> MyWhere<T>(this IEnumerable<T> ie, Func<T, bool> pred)*

*{*

*List<T> result = new List<T>();*

*foreach (T t in ie)*

*if (pred(t))*

*result.Add(t);*

*return result;*

*}*

*public static IEnumerable<T> MyWhere<T>(this IEnumerable<T> ie, Func<T, int, bool> pred)*

*{*

*List<T> result = new List<T>();*

*int index = 0;*

*foreach (T t in ie)*

*if (pred(t, index++))*

*result.Add(t);*

*return result;*

*}*

*}*

*public class App*

*{*

*public static void Main()*

*{*

*List<Product> products = new List<Product> {*

*new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},*

*new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},*

*new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},*

*new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},*

*new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},*

*new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},*

*new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},*

*new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},*

*new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},*

*};*

*var query = products.MyWhere(p => p.ExpiryDate < DateTime.Today);*

*foreach (var p in query)*

*Console.WriteLine(p);*

*Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*");*

*query = products.MyWhere((p, index) => p.ExpiryDate < DateTime.Today && index >= 1);*

*foreach (var p in query)*

*Console.WriteLine(p);*

*}*

*}*

*}*

**Select Metotları**

Bir sorgu sonucunda bir bilginin nasıl bir biçimde elde edileceğine ilişkin bir metottur. İki versiyonu vardır:

public static IEnumerable <Result> Select<T>(this IEnumerable <T> ie, Func<T, Result> selector) {/\*...\*/}

public static IEnumerable <Result> Select<T>(this IEnumerable <T> ie, Func<T, int, Result> selector) {/\*...\*/}

Bu metot ile sorgunun sonucunda istediğimiz alanları içeren bir anonim nesne elde edilebilir. Örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product {

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = products.Where(p => p.ExpiryDate < DateTime.Today).Select(p => new { Name=p.Name, Date=p.ExpiryDate});

foreach (var p in query)

Console.WriteLine(p);

}

}

}

İkinci versiyonundaki index parametresi her elemanın sırasını belirten değerdir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product {

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = products.Where(p => p.ExpiryDate < DateTime.Today).Select((p, index) => new { index, Name=p.Name, Date=p.ExpiryDate});

foreach (var p in query)

Console.WriteLine(p);

}

}

}

**IComparer ve IComparable arayüzleri**

Bu arayüzler genel olarak karşılaştırma işlemi için kullanılan arayüzlerdir. Bir türe ilişkin karşılaştırmanın nasıl yapılacağı bu arayüzler kullanılarak belirlenmektedir. Örneğin, Array sınıfının sort metotları aslında bu arayüzlere ilişkin metotları çağırarak işlem yapar. Sort metodu bir diziyi küçükten büyüğe olacak şekilde sıralar. Neyin küçük neyin büyük olduğuna bu arayüzlere göre karar verir. Sort metodu kendisine geçilecek dizi türünün ICompareable arayüzünü destekleyip desteklemediğine bakar. Destekliyorsa sıralamayı CompareTo metodunun geri dönüş değerine göre yapar:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Person : IComparable<Person>

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime BirthDate { get; set; }

public double Age => (DateTime.Today - BirthDate).TotalDays / 365;

public int CompareTo(Person other)

{

return Age.CompareTo(other.Age);

}

public override string ToString()

{

return $"[{Id}]{Name}:{Age}";

}

}

public class App

{

public static void Main()

{

Person[] people = {

new Person { Id = 2, Name = "Ali", BirthDate = new DateTime(1978, 2, 3) },

new Person { Id = 1, Name = "Veli", BirthDate = new DateTime(1987, 2, 3) }};

Array.Sort(people);

foreach (var p in people)

Console.WriteLine($"{p}");

Console.WriteLine();

}

}

}

IComparable arayüzünü destekleyen bir tür için karşılaştırma işleminin yapılacağı kriterin değiştirilmesi gerektiğinde IComparer arayüzü kullanılabilir. Aşağıdaki örnekte kişiler yaşa göre büyükten küçüğe sıralanmıştır:

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

namespace CSD

{

class Person : IComparable<Person>

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime BirthDate { get; set; }

public double Age => (DateTime.Today - BirthDate).TotalDays / 365;

public int CompareTo(Person other)

{

return Age.CompareTo(other.Age);

}

public override string ToString()

{

return $"[{Id}]{Name}:{Age}";

}

}

class PersonAgeDescComparer : IComparer<Person>

{

public int Compare(Person x, Person y)

{

return y.Age.CompareTo(x.Age);

}

}

public class App

{

public static void Main()

{

Person[] people = {

new Person { Id = 2, Name = "Ali", BirthDate = new DateTime(1978, 2, 3) },

new Person { Id = 1, Name = "Veli", BirthDate = new DateTime(1987, 2, 3) }};

Array.Sort(people, new PersonAgeDescComparer());

foreach (var p in people)

Console.WriteLine($"{p}");

Console.WriteLine();

}

}

}

Aşağıdaki örnekte isme göre artan ve azalan sırada sıralama yapılmıştır:

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

namespace CSD

{

class Person : IComparable<Person>

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime BirthDate { get; set; }

public double Age => (DateTime.Today - BirthDate).TotalDays / 365;

public int CompareTo(Person other)

{

return Age.CompareTo(other.Age);

}

public override string ToString()

{

return $"[{Id}]{Name}:{Age}";

}

}

class PersonNamecComparer : IComparer<Person>

{

private bool m\_asc;

public PersonNamecComparer(bool asc = true)

{

m\_asc = asc;

}

public int Compare(Person x, Person y)

{

return m\_asc ? x.Name.CompareTo(y.Name) : y.Name.CompareTo(x.Name);

}

}

public class App

{

public static void Main()

{

Person[] people = {

new Person { Id = 2, Name = "Ali", BirthDate = new DateTime(1978, 2, 3) },

new Person { Id = 1, Name = "Veli", BirthDate = new DateTime(1987, 2, 13) },

new Person { Id = 3, Name = "Ozan", BirthDate = new DateTime(1976, 12, 3) }};

Array.Sort(people, new PersonNamecComparer());

foreach (var p in people)

Console.WriteLine($"{p}");

Console.WriteLine();

Array.Sort(people, new PersonNamecComparer(false));

foreach (var p in people)

Console.WriteLine($"{p}");

Console.WriteLine();

}

}

}

Aşağıda bir diziyi büyükten küçüğe sıralayan bir program yazılmıştır:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Int32DescComparer : IComparer<int>

{

public int Compare(int x, int y)

{

return y - x;

}

}

public class App

{

public static void Main()

{

int[] a = { -1, -6, 7, -8, 9 };

Array.Sort(a, new Int32DescComparer());

foreach (int val in a)

Console.WriteLine(val);

}

}

}

Burada IComparer arayüzü ile aslında Sort metodu kandırılmıştır. Büyük olan değer küçük gösterilmiştir. Küçükten büyüğe sıralama yapılacağı için, büyük olan dizinin düşük indeksli tarafına (sola doğru) atılır.

**OrderBy ve OrderByDescending Metotları**

Sorgulama sonucunda elde edilen bilginin bir elemana göre küçükten büyüğe (ascending) ya da büyükten küçüğe (descending) dizili olarak elde edilmesini sağlayan metotlardır:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = products.Where(p => p.ExpiryDate < DateTime.Today).OrderBy(p => p.ExpiryDate);

foreach (var p in query)

Console.WriteLine(p);

}

}

}

Aynı programın Ling sorgu sentaksı ile yapılışı:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = from p in products

where p.ExpiryDate < DateTime.Today

orderby p.ExpiryDate

select p;

foreach (var p in query)

Console.WriteLine(p);

}

}

}

Aynı işlemler azalan sırada da yapılabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = products.Where(p => p.ExpiryDate < DateTime.Today).OrderByDescending(p => p.ExpiryDate);

foreach (var p in query)

Console.WriteLine(p);

}

}

}

Linq sorgu sentaksı ile yapılışı:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = from p in products

where p.ExpiryDate < DateTime.Today

orderby p.ExpiryDate descending

select p;

foreach (var p in query)

Console.WriteLine(p);

}

}

}

**ThenBy ve ThenByDescending Metotları**

Order by işleminden sıraya dizilen bilginin başka bir elemana göre de sıraya dizilmesini sağlamak için kullanılan metotlardır. Bu metotlar OrderBy metotlarından sonra çağrılabilir:

Linq metot sentaksı ile ThenBy işlemi şu şekilde yapılabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = products.Where(p => p.ExpiryDate < DateTime.Today).OrderByDescending(p => p.ExpiryDate).ThenBy(p => p.Name);

foreach (var p in query)

Console.WriteLine(p);

}

}

}

Linq sorgu sentaksında then by işlemi aşağıdaki gibi yapılabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = from p in products

where p.ExpiryDate < DateTime.Today

orderby p.ExpiryDate descending, p.Name

select p;

foreach (var p in query)

Console.WriteLine(p);

}

}

}

**Reverse Metodu**

Elde edilen sorguya ilişkin verileri ters-yüz etmek kullanılabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = products.Where(p => p.ExpiryDate < DateTime.Today).OrderByDescending(p => p.ExpiryDate).ThenBy(p => p.Name).Reverse();

foreach (var p in query)

Console.WriteLine(p);

}

}

}

Reverse metodunun linq sorgu sentaksında karşılığı yoktur. Fakat aşağıdaki şekilde kullanılabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = (from p in products

where p.ExpiryDate < DateTime.Today

select p).Reverse();

foreach (var p in query)

Console.WriteLine(p);

}

}

}

**Anahtar Notlar:** IQueryable arayüzü için Reverse metodu kullanılamaz.

**Count ve LongCount Metotları**

Count ve LongCount metotları sorgunun sonucunda elde edilen verinin sayısını verir. Count metodu int türden LongCount metodu da long türden değer döndürür:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2034, 3, 21)},

};

long count = products.Where(p => p.ExpiryDate < DateTime.Today).Count();

Console.WriteLine("Count={0}", count);

}

}

}

Metotların predicate parametreli versiyonları belirli koşula uyan elemanların sayısını elde etmekte kullanılır:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2034, 3, 21)},

};

long count = products.Count(p => p.ExpiryDate < DateTime.Today);

Console.WriteLine("Count={0}", count);

}

}

}

Ya da örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12)},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11)},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21)},

};

var query = products.Where(p => p.ExpiryDate < DateTime.Today);

foreach (var p in query)

Console.WriteLine(p);

int count = query.Count(p => p.Name.StartsWith("m"));

Console.WriteLine("Count={0}", count);

}

}

}

LongCount metotları Count metodunun long geri dönüş değerli versiyonları olarak düşünülebilir.

**Sum Metotları**

Toplama işlemi için kullanılan metotlardır:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 3, 4, 5, 6};

int sum = a.Sum();

Console.WriteLine("Sum={0}", sum);

}

}

}

Sum metotlarının da selector parametreleri versiyonları vardır:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public decimal Price { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}-{3}TL", Id, Name, ExpiryDate.ToShortDateString(), Price);

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12), Price=3.45M},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11), Price=30.45M},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Price=100.45M},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12), Price=3.45M},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11), Price=3.45M},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21), Price=321.45M},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12), Price=123.45M},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11), Price=3.45M},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21), Price=30.45M},

};

var sum = products.Where(p => p.ExpiryDate < DateTime.Today).Sum(p => p.Price);

Console.WriteLine("Sum={0}", sum);

}

}

}

Aşağıdaki örnekte son kullanma tarihi geçmiş ve stokta bulunan ürünlerin toplam fiyatı bulunmuştur:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public decimal Price { get; set; }

public int Stock { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}-{3}TL", Id, Name, ExpiryDate.ToShortDateString(), Price);

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12), Stock = 100, Price=3.45M},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11), Stock = 110, Price=30.45M},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Stock = 1000, Price=100.45M},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12), Stock = 100, Price=3.45M},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11), Stock = 1112, Price=3.45M},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21), Stock = 100, Price=321.45M},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12), Stock = 100, Price=123.45M},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11), Stock = 10, Price=3.45M},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2045, 3, 21), Stock = 3, Price=30.45M},

};

var sum = products.Where(p => p.ExpiryDate < DateTime.Today && p.Stock > 0).Sum(p => p.Price \* p.Stock);

Console.WriteLine("Sum={0}", sum);

}

}

}

**Min ve Max Metotları**

Sorgu sonucunda elde edilen elemanların en küçüğünü ve en büyüğünü döndüren metotlardır. Selector parametreli versiyonları istenilen property elemanına göre karşılaştırmanın yapılmasını sağlar:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public decimal Price { get; set; }

public int Stock { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}-{3}TL", Id, Name, ExpiryDate.ToShortDateString(), Price);

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12), Stock = 100, Price=3.45M},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11), Stock = 110, Price=30.45M},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Stock = 1000, Price=100.45M},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12), Stock = 100, Price=3.45M},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11), Stock = 1112, Price=3.45M},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21), Stock = 100, Price=321.45M},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12), Stock = 100, Price=123.45M},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11), Stock = 10, Price=3.45M},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2045, 3, 21), Stock = 3, Price=30.45M},

};

var max = products.Where(p => p.ExpiryDate < DateTime.Today && p.Stock > 0).Max(p => p.Price);

Console.WriteLine("Max={0}", max);

}

}

}

Ya da örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public decimal Price { get; set; }

public int Stock { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}-{3}TL", Id, Name, ExpiryDate.ToShortDateString(), Price);

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12), Stock = 100, Price=3.45M},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11), Stock = 110, Price=30.45M},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Stock = 1000, Price=100.45M},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12), Stock = 100, Price=3.45M},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11), Stock = 1112, Price=3.45M},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21), Stock = 100, Price=321.45M},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12), Stock = 100, Price=123.45M},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11), Stock = 10, Price=3.45M},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2045, 3, 21), Stock = 3, Price=30.45M},

};

var max = products.Where(p => p.ExpiryDate < DateTime.Today && p.Stock > 0).Max(p => p.Price \* p.Stock);

Console.WriteLine("Max={0}", max);

}

}

}

Min va Max metotları karşılaştıracak türün CompareTo metodunu çağırır. Yani programcı kendi türü için bu işlemi yapmak isterse IComparable arayüzünü desteklemelidir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product : IComparable<Product>

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public decimal Price { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}-{3}TL", Id, Name, ExpiryDate.ToShortDateString(), Price);

}

public int CompareTo(Product p)

{

if (Price > p.Price)

return 1;

if (Price < p.Price)

return -1;

return 0;

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12), Price=3.45M},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11), Price=30.45M},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Price=100.45M},

new Product { Id=4, Name = "minecraft ", ExpiryDate=new DateTime(2015, 3, 12), Price=3.45M},

new Product { Id=5, Name = "minecraft", ExpiryDate=new DateTime(2015, 3, 11), Price=3.45M},

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21), Price=321.45M},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12), Price=123.45M},

new Product { Id=8, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11), Price=3.45M},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21), Price=30.45M},

};

var max = products.Where(p => p.ExpiryDate < DateTime.Today).Min();

Console.WriteLine("Max={0}", max);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product : IComparable<Product>

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public decimal Price { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}-{3}TL", Id, Name, ExpiryDate.ToShortDateString(), Price);

}

public int CompareTo(Product p)

{

return Price.CompareTo(p.Price);

}

}

public class App

{

public static void Main()

{

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate = new DateTime(2015, 3, 12), Price=3.45M},

new Product { Id=2, Name = "minecraft hexxit", ExpiryDate =new DateTime(2015, 3, 11), Price=30.45M},

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Price=100.45M},

new Product { Id=4, Name = "minecraft ", ExpiryDate = new DateTime(2015, 3, 12), Price=3.45M},

new Product { Id=5, Name = "minecraft", ExpiryDate = new DateTime(2015, 3, 11), Price=3.45M},

new Product { Id=6, Name = "grow", ExpiryDate = new DateTime(2023, 3, 21), Price=321.45M},

new Product { Id=7, Name = "mine", ExpiryDate = new DateTime(2015, 3, 12), Price=123.45M},

new Product { Id=8, Name = "mine", ExpiryDate = new DateTime(2015, 3, 11), Price=3.45M},

new Product { Id=9, Name = "grow", ExpiryDate = new DateTime(2023, 3, 21), Price=30.45M},

};

var max = products.Where(p => p.ExpiryDate < DateTime.Today).Max();

Console.WriteLine("Max={0}", max);

}

}

}

**Aggregate Metotları**

Programcının kendi belirlediği işlemlerin yapılması için kullanılabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 3, 4, 5, 6 };

int sum = a.Aggregate((r, elem) => r + elem);

Console.WriteLine("Sum={0}", sum);

}

}

}

Ya da örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 3, 4, 5, 6 };

int sum = a.Aggregate((r, elem) => r \* elem);

Console.WriteLine("Sum={0}", sum);

}

}

}

Aşağıdaki örnekte Complex sınıfının + operator metodu olduğundan Aggregate metodu ile kullanılabilmiştir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Complex

{

public double Re { get; set; }

public double Im { get; set; }

public double Norm => Math.Sqrt(Re \* Re + Im \* Im);

public static explicit operator double(Complex z)

{

return z.Norm;

}

public static Complex operator +(Complex z1, Complex z2)

{

return new Complex { Re = z1.Re + z2.Re, Im = z1.Im + z2.Im };

}

//...

public override string ToString()

{

return $"{Re} + i{Im}";

}

}

public class App

{

public static void Main()

{

List<Complex> complexNumbers = new List<Complex> {

new Complex { Re = 3},

new Complex { Re = -5.6, Im = 4.3},

new Complex { Re = -1.2, Im = 7.1},

};

var z = complexNumbers.Aggregate((r, z) => r + z);

Console.WriteLine(z);

Console.WriteLine((double)z);

}

}

}

**Union, Intersect ve Except Metotları**

Bu operatörler birleşim, kesişim ve fark küme işlemlerinin karşılıklarıdır. İki collection’ın birleşimini, kesişimini, farklarını elde etmek için kullanılır. Örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 6, 7 };

List<int> numbers = new List<int> { 8, 8, 9, 10 };

var result = a.Union(numbers);

foreach (var val in result)

Console.WriteLine(val);

}

}

}

Programcının yazdığı türler için bir elemanın tek olması durumu neye göre belirlenecektir? Bunun için birçok yöntem tercih edilebilir. Bu metotlar karşılaştırma işlemi için object sınıfın Equals ve GetHashCode metotlarını çağırmaktadır. Programcı bu metotları override ederek aynı olma kriterlerini kendi yazdığı tür için belirleyebilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public decimal Price { get; set; }

public override bool Equals(object obj)

{

return this.Id == ((Product)obj).Id;

}

public override int GetHashCode()

{

return this.Id;

}

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}-{3}TL", Id, Name, ExpiryDate.ToShortDateString(), Price);

}

}

public class App

{

public static void Main()

{

Product product = new Product { Id = 2, Name = "minecraft", ExpiryDate = new DateTime(2015, 3, 11), Price = 30.45M };

List<Product> products1 = new List<Product> {

new Product { Id=1, Name = "minecraft test", ExpiryDate=new DateTime(2015, 3, 12), Price=3.45M},

product,

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Price=100.45M},

};

List<Product> products2 = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12), Price=3.45M},

new Product { Id=10, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11), Price=30.45M},

new Product { Id=34, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Price=100.45M},

product,

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21), Price=321.45M},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12), Price=123.45M},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11), Price=3.45M},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21), Price=30.45M},

};

var products = products1.Union(products2);

foreach (Product p in products)

Console.WriteLine(p);

}

}

}

Programcının elinde sınıfın kodları yoksa ya da metotları override etmek istemezse bu durumda IEqualityComparer arayüzünü destekleyen bir sınıf yazarak bu sınıftan bir nesneyi Union, Intersection ya da Except metotlarına geçebilir ve karşılaştırmanın yapılmasını sağlayabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public decimal Price { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}-{3}TL", Id, Name, ExpiryDate.ToShortDateString(), Price);

}

}

public class ProductComparer : IEqualityComparer<Product> {

public bool Equals(Product x, Product y)

{

return x.Id == y.Id;

}

public int GetHashCode(Product obj)

{

return obj.Id;

}

}

public class App

{

public static void Main()

{

Product product = new Product { Id = 2, Name = "minecraft", ExpiryDate = new DateTime(2015, 3, 11), Price = 30.45M };

List<Product> products1 = new List<Product> {

new Product { Id=1, Name = "minecraft test", ExpiryDate=new DateTime(2015, 3, 12), Price=3.45M},

product,

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Price=100.45M},

};

List<Product> products2 = new List<Product> {

new Product { Id=1, Name = "minecraft tekkit", ExpiryDate=new DateTime(2015, 3, 12), Price=3.45M},

new Product { Id=10, Name = "minecraft hexxit", ExpiryDate=new DateTime(2015, 3, 11), Price=30.45M},

new Product { Id=34, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Price=100.45M},

product,

new Product { Id=6, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21), Price=321.45M},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 12), Price=123.45M},

new Product { Id=7, Name = "mine", ExpiryDate=new DateTime(2015, 3, 11), Price=3.45M},

new Product { Id=9, Name = "grow", ExpiryDate=new DateTime(2016, 3, 21), Price=30.45M},

};

var products = products1.Union(products2, new ProductComparer());

foreach (Product p in products)

Console.WriteLine(p);

}

}

}

Diğer bir yöntem de sınıf yerine yapı kullanmaktır. Ancak bu durum her zaman mümkün olmayabilir. Çünkü programcının bildirdiği tür yapı olamayacak kadar fazla yer kaplayabilir, ya da bu tür türetme ile de kullanılacak olabilir. Yani yapı olması gereken durumlar için tercih edilebilir.

***Anahtar Notlar:*** *Bilindiği gibi yapılar kategori olarak değer türlerine ilişkindir. Yapılar, ValueType isimli abstract bir sınıftan türetilmiştir. Bu sınıfın Equals metodu karşılıklı elemanların eşitlik karşılaştırmasını yapar. Dolayısıyla bir yapının karşılıklı elemanları aynı ise yapı değişkenleri eşittir denebilir. Benzer şekilde yapılar için GetHashCode metotları sınıflarda olduğu gibi bellek adresine ilişkin bir değeri döndürmez. Aynı elemanlara sahip yapılar için aynı hash kodu üretilir.*

**Zip Metodu**

Zip metodu .NET 4.0 ile birlikte eklenmiş bir metottur. Bu metot ile iki IEnumerable referansına ilişkin bilgi birleştirilebilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 34, 6, 67, 35, 1, 78 };

List<string> cities = new List<string>{ "istanbul", "ankara", "zonguldak", "izmir", "adana" };

var result = a.Zip(cities, (p, c) => p + "-" + c);

foreach (var s in result)

Console.WriteLine(s);

}

}

}

Bu metot ile farklı türden collection kombine edilerek istenilen şekilde farklı türden bir collection da elde edilebilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class CityInfo

{

public int Plate { get; set; }

public string Name { get; set; }

public override string ToString()

{

return string.Format("{0}-{1}", Plate, Name);

}

}

public class App

{

public static void Main()

{

int[] a = { 34, 6, 67, 35, 1 };

string[] cities = { "istanbul", "ankara", "zonguldak", "izmir", "adana" };

var result = a.Zip(cities, (val, s) => new CityInfo { Name = s, Plate = val });

foreach (var s in result)

Console.WriteLine(s);

}

}

}

Ya da örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class CityInfo

{

public int Plate { get; set; }

public string Name { get; set; }

public override string ToString()

{

return string.Format("{0}-{1}", Plate, Name);

}

}

public class App

{

public static void Main()

{

int[] a = { 34, 6, 67, 35, 1, 67};

string[] cities = { "istanbul", "ankara", "zonguldak", "izmir", "adana" };

var result = a.Zip(cities, (val, s) => new CityInfo { Name = s, Plate = val });

foreach (var s in result)

Console.WriteLine(s);

}

}

}

**Range Metodu**

Bu metot birinci parametresi ile aldığı başlangıç değerinden itibaren ikinci parametresi ile aldığı sayı kadar int türden peşpeşe gelen sayılardan oluşan bir collection referansı döndürür.

public static IEnumerable<int> Range(int start, int count)

Bu metot ile istenilen aralıkta değer üretilebilir. Dikkat edilirse bu metot extension metot olarak bildirilmemiştir. Doğrudan Enumerable sınıf ismi ile çağrılmaktadır. Bu metotlar yalnızca “Linq to object” için geçerlidir. Örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

IEnumerable<int> coll = Enumerable.Range(-10, 20);

Console.WriteLine(coll.GetType().Name);

foreach (int val in coll)

Console.WriteLine(val);

}

}

}

Bu metot ile bazı işlemler fazla kod yazmadan da yapılabilir. Örneğin faktoriyel işlemi bu metot ile kolaylıkla kodlanabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static int Factorial(int n)

{

return Enumerable.Range(1, n).Aggregate(1, (r, val) => r \* val);

}

public static void Main()

{

for (int n = 0; n <= 10; ++n)

Console.WriteLine($"{n}!={Factorial(n)}");

}

}

}

Örneğin 1 den n'e kadar olan sayıların toplamı da bu şekilde bulunabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static int Sum(int n)

{

return Enumerable.Range(1, n).Sum();

}

public static void Main()

{

Console.WriteLine(Sum(100));

}

}

}

**Repeat Metodu**

Bu metot birinci parametresi aldığı değerden, ikinci parameteresi ile aldığı değer kadar aynı elemandan içeren bir IEnumerable referansı döndürür. Örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

IEnumerable<bool> status = Enumerable.Repeat(true, 10);

foreach (bool b in status)

Console.WriteLine(b);

}

}

}

Bu metot da eklenti metot değildir. Repeat metodu ile her tür kullanılabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

IEnumerable<char> status = Enumerable.Repeat('c', 10);

foreach (var b in status)

Console.WriteLine(b);

}

}

}

**Empty Metodu**

Bu metot herhangi bir türden boş bir IEnumerable referansı döndürür. Örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

IEnumerable<int> coll = Enumerable.Empty<int>();

Console.WriteLine(coll.Count());

}

}

}

**Any Metotları**

Bu metotlar bir IEnumerable’ın boş olup olmadığını ya da belirlenen koşula uygun eleman içerip içermediğini test eder:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { -1, 2, 3, 5 };

if (a.Any(val => val < 0))

Console.WriteLine("var");

else

Console.WriteLine("yok");

}

}

}

Any metodunun tek parametreli (this parametreli) versiyonu collection boş ise false herhangi bir eleman içeriyorsa true değerini döndürür. Bu metot ile bir collection'ın boş olup olmadığı testi kolayca yapılabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { -1, 2, 3, 5 };

if (a.Any())

Console.WriteLine("Dolu");

else

Console.WriteLine("Boş");

}

}

}

**All Metodu**

Bu metot bir IEnurable içerisindeki elemanların tümünün birden belirlenen koşula uygun olup olmadığını test eder. Eğer bir eleman bile bu koşula uymuyorsa false değeri ile geri döner. Örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { -1, 2, 3, 5 };

if (a.All(val => val > 0))

Console.WriteLine("Hepsi Pozitif");

else

Console.WriteLine("Pozitif olmayan eleman(lar) da var");

}

}

}

All metodunun ilginç bir davranışı vardır. Bu metot boş bir collection için true değerini döndürür:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = {};

if (a.All(val => val > 0))

Console.WriteLine("Doğru");

else

Console.WriteLine("Yanlış");

}

}

}

Bir collection’ın boş olup olmamasına göre All metodunun bir kullanımı aşağıdaki gibi olabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { }; // { -1, 2, 3, 5 };

if (a.Any())

{

if (a.All(val => val > 0))

Console.WriteLine("Hepsi Pozitif");

else

Console.WriteLine("Pozitif olmayan eleman(lar) da var");

}

else

Console.WriteLine("Dizi boş");

}

}

}

**Contains Metotları**

Bu metotlar bir collection içerisinde bir elemanın olup olmadığını test eder. Metodun programcının belirlediği bir tür için IEqualityComparer arayüzü türünden parametreli bir versiyonu vardır.

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public decimal Price { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}-{3}TL", Id, Name, ExpiryDate.ToShortDateString(), Price);

}

}

public class ProductComparer : IEqualityComparer<Product>

{

public bool Equals(Product x, Product y)

{

return x.Id == y.Id;

}

public int GetHashCode(Product p)

{

return p.Id;

}

}

public class App

{

public static void Main()

{

Product product = new Product { Id = 2, Name = "minecraft", ExpiryDate = new DateTime(2015, 3, 11), Price = 30.45M };

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft test", ExpiryDate=new DateTime(2015, 3, 12), Price=3.45M},

product,

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Price=100.45M},

};

if (products.Contains(new Product { Id = 2}, new ProductComparer()))

Console.WriteLine("var");

else

Console.WriteLine("Yok");

}

}

}

**Take Metodu**

Sorgu sonucunda elde edilen collection içerisinden istenildiği kadar elemanı almak için kullanılır:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public decimal Price { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}-{3}TL", Id, Name, ExpiryDate.ToShortDateString(), Price);

}

}

public class ProductComparer : IEqualityComparer<Product>

{

public bool Equals(Product x, Product y)

{

return x.Id == y.Id;

}

public int GetHashCode(Product obj)

{

return obj.Id;

}

}

public class App

{

public static void Main()

{

Product product = new Product { Id = 2, Name = "minecraft", ExpiryDate = new DateTime(2016, 3, 27), Price = 30.45M };

List<Product> products = new List<Product> {

new Product { Id=1, Name = "minecraft test", ExpiryDate=new DateTime(2015, 3, 12), Price=3.45M},

product,

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Price=100.45M},

};

var query = products.Where(p => p.ExpiryDate < DateTime.Today).Take(2);

foreach (var p in query)

Console.WriteLine(p);

}

}

}

Bu metodun aldığı parametreden daha az sayıda eleman varsa bu durumda o kadar elemanı içeren bir IEnumerable referansı döner. Bu metot parametresi ile aldığı index değeri pozitif değilse veya sorgu sonucunda hiç bir eleman gelmemişse boş bir IEnumerable döndürür.

**TakeWhile Metotları**

Bu metotlar predicate parametresine ilişkin koşula uymayan elemanı görene kadar tüm elemanları bir collectionda toplar ve o collection için IEnuerable referansını döndürür:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public decimal Price { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}-{3}TL", Id, Name, ExpiryDate.ToShortDateString(), Price);

}

}

public class ProductComparer : IEqualityComparer<Product>

{

public bool Equals(Product x, Product y)

{

return x.Id == y.Id;

}

public int GetHashCode(Product obj)

{

return obj.Id;

}

}

public class App

{

public static void Main()

{

Product product = new Product { Id = 2, Name = "minecraft", ExpiryDate = new DateTime(2016, 3, 27), Price = 30.45M };

List<Product> products = new List<Product> {

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Price=100.45M},

new Product { Id=1, Name = "minecraft test", ExpiryDate=new DateTime(2015, 3, 12), Price=3.45M},

product,

};

var query = products.TakeWhile((p, index) => p.ExpiryDate < DateTime.Today);

foreach (var p in query)

Console.WriteLine(p);

}

}

}

Metodun diğer versiyonu int parametreli bir predicate alır. Bu parametre her adımda index değerini belirtir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public decimal Price { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}-{3}TL", Id, Name, ExpiryDate.ToShortDateString(), Price);

}

}

public class ProductComparer : IEqualityComparer<Product>

{

public bool Equals(Product x, Product y)

{

return x.Id == y.Id;

}

public int GetHashCode(Product obj)

{

return obj.Id;

}

}

public class App

{

public static void Main()

{

Product product = new Product { Id = 2, Name = "minecraft", ExpiryDate = new DateTime(2016, 3, 27), Price = 30.45M };

List<Product> products = new List<Product> {

new Product { Id=3, Name = "growtopia", ExpiryDate=new DateTime(2016, 3, 21), Price=100.45M},

new Product { Id=1, Name = "minecraft test", ExpiryDate=new DateTime(2015, 3, 12), Price=3.45M},

product,

};

var query = products.TakeWhile((p, index) => p.ExpiryDate < DateTime.Today && index < 1);

foreach (var p in query)

Console.WriteLine(p);

}

}

}

**Skip Metodu**

Bu metot parametresi aldığı değer kadar elemanı collection içerisinde atlar ve geri kalanı verir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 3, -3, 6, 7 };

var coll = a.Skip(3);

foreach (var val in coll)

Console.WriteLine(val);

}

}

}

Aşağıdaki kodda dizinin elemanlarının tekrarlanmadığı bir collection elde edilmiştir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 3, 3, -3, 7, 6, 7 };

var coll = a.Take(3).Union(a.Skip(3));

foreach (var val in coll)

Console.WriteLine(val);

}

}

}

**SkipWhile Metotları**

Bu metotlar predicate parametresine ilişkin koşula uymayan ilk elemanı görene kadar tüm elemanları atlar ve gördükten sonraki eleman ilk gördüğü elemanı içeren bir IEnumerable referansı döndürür:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 3, -3, 2, 6, 7 };

var coll = a.SkipWhile(val => val % 2 != 0);

foreach (var val in coll)

Console.WriteLine(val);

}

}

}

**First Metotları**

Bu metot collection içerisindeki ilk elemanı verir. Eğer collection içerisinde eleman yoksa exception fırlatır.

Örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 3, -3, 21, 61};

try

{

int first = a.Where(val => val % 2 == 0).First();

Console.WriteLine(first);

}

catch (InvalidOperationException ex) {

Console.WriteLine("Belirlenen koşula uygun eleman bulunamadı");

}

}

}

}

Bu metodun predicate parametreli versiyonu ile doğrudan koşul da verilebilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 3, -3, 21, 61};

try

{

int first = a.First(val => val % 2 == 0);

Console.WriteLine(first);

}

catch (InvalidOperationException ex) {

Console.WriteLine("Belirlenen koşula uygun eleman bulunamadı");

}

}

}

}

**FirstOrDefault Metotları**

Bu metot ilk elemanı verir. Eğer collection boş ise collection hangi tür içinse o türe ilişkin default değeri verir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 3, -3, 21, 61};

int first = a.FirstOrDefault(val => val % 2 == 0);

Console.WriteLine(first);

}

}

}

***Anahtar Notlar:*** *Her türün bir default değeri vardır. Referans türlerinin default değeri null'dır. Bu durumda programcı referans türleri ile çalışacaksa XXXOrDefault metotlarının geri dönüş değerleri için null kontrolü yapmalıdır.*

**Last Metotları**

Bu metot son elemanı elde etmekte kullanılır. Son elemanı vermesi dışında First metoduyla aynı davranıştadır:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 3, -3, 2, 6 };

try

{

int first = a.Where(val => val % 2 == 0).Last();

Console.WriteLine(first);

}

catch (InvalidOperationException)

{

Console.WriteLine("Belirlenen koşula uygun eleman bulunamadı");

}

}

}

}

**LastOrDefaultMetotları**

Bu metot da son elemanı elde etmekte kullanılır. Son elemanı vermesi dışında FirstOrDefault metotlarıyla aynı davranıştadır.

**Single Metotları**

Bu metotlar bir collection içerisindeki tekil (unique) elemanı elde etmekte kullanılır. Predicate parametreli versiyonu o koşula uygun tek bir tane eleman varsa onu verir. Yoksa ya da koşula uyan birden fazla eleman varsa exception fırlatır:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 3, -3, 21, 6 };

try

{

int val = a.Single(val => val % 2 == 0);

Console.WriteLine(val);

}

catch (InvalidOperationException)

{

Console.WriteLine("Belirlenen koşula uygun tekil eleman bulunamadı");

}

}

}

}

Parametresiz versiyonu (sadece eklenti parametreli versiyonu) uygulanan collectionın bir tane elemanı varsa o elemanı verir, yoksa exception fırlatır:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 3, -3, 21, 6, 2 };

try

{

int val = a.Where(val => val % 2 == 0).Single();

Console.WriteLine(val);

}

catch (InvalidOperationException)

{

Console.WriteLine("Belirlenen koşula uygun eleman bulunamadı");

}

}

}

}

**SingleOrDefaultMetotları**

Bu metot Single metotlarının exception fırlatmayan, türün default değerini döndüren metotlardır.

**ElementAt Metodu**

Bu metot herhangi bir indexteki elemanı almak için kullanılır. Eğer index değeri pozitif ya da negatif bakımdan sınırlar dışında kalıyorsa ArgumentOutOfRangeException nesnesi fırlatılır:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 3, -3, 21, 6 };

try

{

int elem = a.Where(val => val % 2 == 0).ElementAt(2);

Console.WriteLine(elem);

}

catch (ArgumentOutOfRangeException)

{

Console.WriteLine("Index Taşması");

}

}

}

}

**ElementAtOrDefault Metodu**

Bu metot ElementAt metodunun exception fırlatmayan versiyonudur. Eğer eleman yoksa ya da sınır dışında bir index verilmişse o elemana ilişkin default değer döndürülür.

**DefaultIsEmpty Metotları**

Eğer bir collection ise bu metot ile default değerler elde edilebilir. Bu metot eğer collection boş ise yine bir collection referansı döndürür fakat bu collection tek elemanlıdır:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 3, -3, 21, 6 };

try

{

var coll = a.Where(val => val % 11 == 0).DefaultIfEmpty(-11);

foreach (var val in coll)

Console.WriteLine(val);

}

catch (ArgumentOutOfRangeException)

{

Console.WriteLine("Index Taşması");

}

}

}

}

Bu metot örneğin FirstOrDefault metodu yerine programcının belirlediği default değer için aşağıdaki gibi kullanılabilir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 3, -3, 21, 6 };

var elem = a.Where(val => val % 11 == 0).DefaultIfEmpty(10).First();

Console.WriteLine(elem);

}

}

}

Yukarıdaki örnekte DefaultOrEmpty metodunun parametresiz versiyonu çağrılırsa FirstOrDefault ile aynı şey yapılmış olur

**Concat Metodu**

Bu metot aynı türden iki IEnumerable referansına ilişkin collectionları birleştirmek için kullanılır. Burada aynı tür kavramı IEnumerable arayüzünün açılımlarının aynı olması anlamındadır. Concat metodu aşağıdaki gibidir:

public IEnumerable<T> Concat(this IEnumerable<T> ie, IEnumerable<T> a)

Örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 3, 4 };

List<int> list = new List<int>() { 5, 6, 4, 7, 8 };

IEnumerable<int> result = a.Concat(list);

foreach (int val in result)

Console.WriteLine(val);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 3, 4 };

List<int> list = new List<int>() { 1, 2, 3, 4 };

Console.WriteLine(a.SequenceEqual(list));

}

}

}

IEqualityComparer arayüzü ile karşılaştırma kriteri belirlenebilir:

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

namespace CSD

{

class MyEqualityComparer : IEqualityComparer<string>

{

public bool Equals(string x, string y)

{

return x.ToLower() == y.ToLower();

}

public int GetHashCode(string s)

{

return s.ToLower().GetHashCode();

}

}

public class App

{

public static void Main()

{

string[] cities = { "ankara", "adana", "bolu", "zonguldak" };

List<string> clist = new List<string> { "ANKARA", "ADANA", "BOLU", "ZONGULDAK" };

Console.WriteLine(cities.SequenceEqual(clist, new MyEqualityComparer()));

}

}

}

**OfType ve Cast Metotları**

Bu metotlar bir türde açılmış collectionı’ başka bir türden collection yapmak için kullanılmaktadır. İkisinin arasında çok küçük bir fark vardır. OfType dönüştürülemeyen elemanları görmezden gelir, Cast ise InvalidOperationException nesnesi fırlatır:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

object[] cities = { "ankara", "istanbul", "izmir"};

IEnumerable<string> a = cities.OfType<string>();

foreach (var s in a)

Console.WriteLine(s.ToUpper());

}

}

}

Örneğin:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Shape {

//...

}

class Rectangle : Shape {

//...

}

class Triangle : Shape {

}

class Line : Shape {

}

public class App

{

public static void Main()

{

List<Shape> shapes = new List<Shape> { new Line(), new Rectangle(), new Triangle(), new Rectangle(), new Triangle(), new Line(), new Rectangle() };

var rectangles = shapes.OfType<Rectangle>();

Console.WriteLine(rectangles.Count());

}

}

}

Bu metotlar her eleman için tür dönüştürme operatörü uygular. Yani downcasting veya boxing uygulanır. Tabii ki OfType metodu dinamik türü kapsamayan türleri görmezden gelir:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

object[] objs = { 10, 20L, "ankara", 30, 40, 50L, "istanbul" };

var longs = objs.OfType<long>();

foreach (var val in longs)

Console.WriteLine(val);

}

}

}

***Anahtar Notlar:*** *C#'da boxing ve unboxing işlemlerinde özellike unboxing yapılırken hedef türün farklı olması haksız dönüşümdür:*

***using System;***

*using System.Collections.Generic;*

*using System.Linq;*

*namespace CSD*

*{*

*public class App*

*{*

*public static void Main()*

*{*

*int val = 10;*

*object o = val;*

*long b = (long)o; //exception: Haksız dönüşüm*

*Console.WriteLine(b);*

*}*

*}*

*}*

*using System;*

*using System.Collections.Generic;*

*using System.Linq;*

*namespace CSD*

*{*

*public class App*

*{*

*public static void Main()*

*{*

*int val = 10;*

*object o = val;*

*short b = (short)(int)o;*

*Console.WriteLine(b);*

*}*

*}*

*}*

Benzer şekilde Cast metodu da kullanılabilir. Bu durumda exception oluşur:

using System;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Shape

{

//...

}

class Rectangle : Shape

{

//...

}

class Triangle : Shape

{

}

class Line : Shape

{

}

public class App

{

public static void Main()

{

List<Shape> shapes = new List<Shape> { new Line(), new Rectangle(), new Triangle(), new Rectangle(), new Triangle(), new Line(), new Rectangle() };

var rectangles = shapes.Cast<Rectangle>();

Console.WriteLine(rectangles.Count());

}

}

}

OfType metodu basit bir şekilde yazılabilir (Detaylar atlanmıştır):

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class Shape

{

//...

}

class Rectangle : Shape

{

//...

}

class Triangle : Shape

{

}

class Line : Shape

{

}

public class App

{

public static void Main()

{

List<Shape> shapes = new List<Shape> { new Line(), new Rectangle(), new Triangle(), new Rectangle(), new Triangle(), new Line(), new Rectangle() };

var rectangles = shapes.MyOfType<Rectangle>();

Console.WriteLine(rectangles.Count());

}

}

static class MyEnumerable {

public static IEnumerable<T> MyOfType<T>(this IEnumerable ie)

{

List<T> list = new List<T>();

foreach (var val in ie) {

try

{

list.Add((T)val);

}

catch (InvalidCastException)

{

}

}

return list;

}

}

}

**ToArray metodu**

Bu metot ilgili IEnumerable referansına ilişkin collection dan bir dizi elde etmekte kullanılmaktadır:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };

int[] evens = a.Where(val => val % 2 == 0).OrderByDescending(val => val).Take(4).ToArray();

foreach (int val in evens)

Console.WriteLine(val);

}

}

}

**ToList Metodu**

Bu metot ilgili Ienumerable referansına ilişkin collection dan bir dinamik büyüyen dizi (List) elde etmekte kullanılmaktadır:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };

List<int> evens = a.Where(val => val % 2 == 0).OrderByDescending(val => val).Take(4).ToList();

foreach (int val in evens)

Console.WriteLine(val);

}

}

}

List sınıfının bir çok yararlı metodu vardır. Bunlardan en çok kullanılanlarından biri ForEach metodudur. Bu metot foreach döngüsünü temsil eden metottur:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };

a.Where(val => val % 2 == 0)

.OrderByDescending(val => val)

.Take(4)

.ToList()

.ForEach(val => Console.WriteLine(val));

}

}

}

List sınıfının FindLast metodu belirli bir koşula uyan son elemanı elde etmekte kullanılır. Belirlenen koşula uygun hiç eleman yoksa açıldığı türe ilişkin default değeri döndürür:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };

var val = a.Where(val => val % 2 == 0)

.Take(4)

.ToList()

.FindLast(val => val > 3);

Console.WriteLine(val);

}

}

}

List sınıfının FindAll metodu belirli koşula uyan tüm elemanları bir List ile elde etmekte kullanılır:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };

a.Where(val => val % 2 == 0)

.Take(4)

.ToList()

.FindAll(val => val > 5)

.ForEach(val => Console.WriteLine(val));

}

}

}

List sınıfının Find metodu belirlenen koşula uygun ilk elemanı döndürmektedir:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };

var val = a.Where(val => val % 2 == 0)

.Take(4)

.ToList()

.Find(val => val > 3);

Console.WriteLine(val);

}

}

}

List sınıfının Exists metodu belirli koşula uygun olan elemanın varlığını test etmek için kullanılır:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };

var status = a.Where(val => val % 2 == 0)

.Take(4)

.ToList()

.Exists(val => val > 30);

Console.WriteLine(status);

}

}

}

ToArray ve ToList metotları collection içerisinden o anki kopyayı çıkarmaktadır (snapshot). Şüphesiz içsel olarak gerektiğinde kopy çıkarma (copy on write) yöntemini kullanabilir. Listede ya da ana collaction’da bir değişiklik diğerlerini etkilemez:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

int[] a = { 1, 2, 3, 4, 5 };

List<int> evens = a.ToList();

a[0] = 20;

evens.ForEach(val => Console.WriteLine(val));

}

}

}

Aşağıdaki örnekte [1, n] aralığındaki sayılar ekrana basılmaktadır:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class App

{

public static void Main()

{

Console.Write("Bir sayı giriniz:");

int n = int.Parse(Console.ReadLine());

Enumerable.Range(1, n).ToList().ForEach(val => Console.WriteLine(val));

}

}

}

Aşağıdaki örnekte bir sayının asal olup olmadığını test eden metot yazılmıştır:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

class NumberUtil {

public static bool IsPrime(int val)

{

if (val <= 1)

return false;

if (val == 2)

return true;

int halfLen = val / 2;

return Enumerable.Range(2, halfLen).All(i => val % i != 0);

}

}

public class App

{

public static void Main()

{

Enumerable.Range(0, 101).ToList().ForEach(val => {

if (NumberUtil.IsPrime(val))

Console.Write($"{val} ");

});

Console.WriteLine();

Console.WriteLine(NumberUtil.IsPrime(1000003));

}

}

}

**AsEnumerable Metodu**

Bu metot bir IEnumerable collection dan yine bir IEnumerable elde etmek için kullanılır. Programcının kendi collection sınıfını ve Enumerable sınıfı ile aynı isimde metotları olan başka bir sınıf yazması durumunda tercih edilebilir:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace CSD

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime ExpiryDate { get; set; }

public override string ToString()

{

return string.Format("[{0}]-{1}-{2}", Id, Name, ExpiryDate.ToShortDateString());

}

}

public static class ProductListExtensions

{

public static ProductList Where(this ProductList list, Func<Product, bool> predicate)

{

Console.WriteLine("ProductList.Where");

ProductList result = new ProductList();

foreach (Product p in list)

if (predicate(p))

result.Add(p);

return result;

}

}

public class ProductList : List<Product>

{

public ProductList()

{ }

public ProductList(params Product[] products)

{

if (products == null)

throw new ArgumentNullException("Null parameter not allowed");

foreach (Product p in products)

if (p != null)

base.Add(p);

}

new public bool Add(Product p)

{

if (p == null)

return false;

base.Add(p);

return true;

}

}

public class App

{

public static void Main()

{

ProductList list = new ProductList(new Product { Name = "Çikolata", ExpiryDate = DateTime.Today, Id = 1 }, new Product { Name = "Gofret", ExpiryDate = DateTime.Today, Id = 2 }, new Product { Name = "Bal", ExpiryDate = DateTime.Today, Id = 3 });

var newList = list.AsEnumerable();

var query = newList.Where(p => p.Id > 1);

query.ToList().ForEach(p => Console.WriteLine(p));

}

}

}