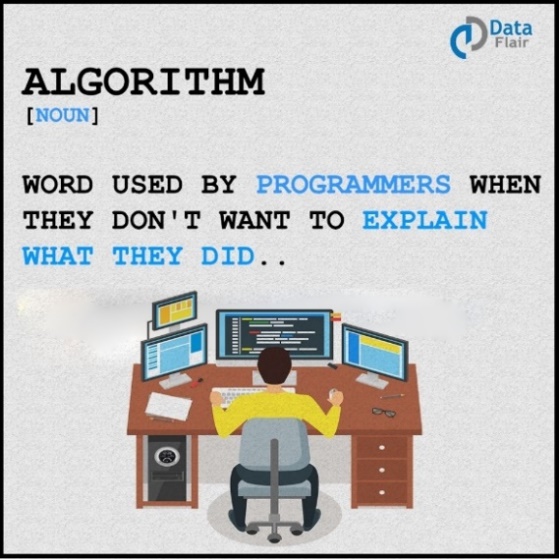
# LINQ

|  |  |
| --- | --- |
| **Læringsmål** | Du kan:   * **1Pf1**: anvende centrale metoder til at specificere og konstruere algoritmer […] * **1Pf2**: anvende centrale faciliteter i programmeringssproget til realisering af algoritmer, designmønstre, abstrakte datatyper, datastrukturer, designmodeller og brugergrænseflader. |
| **Forventet læringsudbytte [SOLO]** | Programmeringssporet:   * [Unistrukturel] Du kan genkende enkelte kendetegn ved **Enumerable-klassen** * [Unistrukturel] Du kan genkende enkelte kendetegn ved **LINQ** * [Unistrukturel] Du kan genkende enkelte kendetegn ved **LINQ Methods** |
| **Din forberedelse** | * Overview – LINQ (video: 7:57; se i planen i ItsLearning) * Sjov med LINQ (video: 30:53; se i planen i ItsLearning)   Følgende links viser eksempler for C#. Ignorér eksemplerne for Visual Basic (VB.NET).   * [What is LINQ](http://www.tutorialsteacher.com/linq/what-is-linq) * [Why LINQ?](http://www.tutorialsteacher.com/linq/why-linq) * [LINQ API](http://www.tutorialsteacher.com/linq/linq-api) * [LINQ Query Syntax](https://www.tutorialsteacher.com/linq/linq-query-syntax) * [LINQ Method Syntax](http://www.tutorialsteacher.com/linq/linq-method-syntax) * [Lambda Expressions](http://www.tutorialsteacher.com/linq/linq-lambda-expression) * [Standard Query Operators](http://www.tutorialsteacher.com/linq/linq-standard-query-operators) * [Where Filtering Operator](http://www.tutorialsteacher.com/linq/linq-filtering-operators-where) * [OrderBy Sorting Operator](https://www.tutorialsteacher.com/linq/linq-sorting-operators-orderby-orderbydescending) * [Sum Aggregation Operator](http://www.tutorialsteacher.com/linq/linq-aggregation-operator-sum) * Læs om LINQ   + [Where](https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.where)   + [ToList](https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.tolist)   + [FirstOrDefault](https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.firstordefault)   + [Count](https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.count)   + [Contains](https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.contains) |

LINQ har to forskellige syntakser, *LINQ Query syntaks* og *LINQ Method syntaks*. Du ser i denne opgave kun på den sidste syntaks *LINQ Method syntaks* – LINQ Query syntaksen er reelt set kun en mere kompakt SQL-lignende version af LINQ Method syntaksen, og du er velkommen til selv at eksperimentere med den på egen hånd.



# Øvelse 1: Terminologi

Del teamet op i mindre grupper, og brug **Ordet rundt** til at reflektere over begreberne ”Enumerable”, ”Lambda udtryk”, ”LINQ” og ”LINQ Method Syntax”.  
Sørg for at alle får mulighed for at tale.

*Tidsramme: 20 minutter*

**Benyt parprogrammering til alle følgende øvelser.**

# Øvelse 2: LINQ og Lambda-udtryk

De to statiske .NET-klasser *Enumerable* og *Queryable* er centrale for LINQ API’et og indeholder en bred vifte af extension-metoder, som kombineret med lambda-udtryk er en kraftfuld og kompakt måde at håndtere klasser, som implementerer de to interfaces *IEnumerable* og *IQueryable*.

Du skal i denne øvelse arbejde med nogle få extension-metoder *Sum()*, *Where()* og *OrderBy()* fra *Enumerable*-klassen.

I de efterfølgende øvelser benyttes Ex50-LINQ.zip – find den i materialemappen på GitLab.

## Øvelse 2.1: Anvendelse af Enumerable.Sum()

Metoden *GetValueOfProducts()* i Order-klassen beregner den samlede værdi af alle produkter i en ordre.

Udfør følgende:

* Refaktorér koden i denne metode, så den anvender extension-metoden *Enumerable.Sum()* og et passende lambda-udtryk i stedet for en foreach-løkke til at beregne summen af produkternes værdi.
* Kør testen, og tilret om nødvendigt.

## Øvelse 2.2: Anvendelse af Enumerable.Where()

Udvid Product-klassen med to nye properties *AvailableFrom* og *AvailableTo* af typen *DateTime*, så klassen kommer til at se ud som forneden:

public class Product

{

public string Name { get; set; }

public double Value { get; set; }

public DateTime AvailableFrom { get; set; }

public DateTime AvailableTo { get; set; }

}

hvor:

* **AvailableFrom** angiver fra hvilken dato, produktet er tilgængeligt
* **AvailableTo** angiver til hvilken dato, produktet er tilgængeligt

Udvid *InitializeTest()*-metoden i UnitTest1.cs, så de to nye properties bliver initialiseret på følgende måde:

[TestInitialize]

public void InitializeTest()

{

order = new Order();

order.AddProduct(new Product { Name = "Mælk",

Value = 10.0,

AvailableFrom = new DateTime(2018, 3, 1),

AvailableTo = new DateTime(2018, 3, 5)

});

order.AddProduct(new Product { Name = "Smør",

Value = 15.0,

AvailableFrom = new DateTime(2018, 3, 3),

AvailableTo = new DateTime(2018, 3, 4)

});

order.AddProduct(new Product { Name = "Pålæg",

Value = 20.0,

AvailableFrom = new DateTime(2018, 3, 4),

AvailableTo = new DateTime(2018, 3, 7)

});

}

Udfør følgende:

* Overload *GetValueOfProducts()* med metoden *GetValueOfProducts(DateTime date)* i Order-klassen, hvor man i parameteren *date* kan angive den dato, produktværdien skal beregnes for.   
  I den nye overloadede metode, benyt *Enumerable.Where()* med et passende lambda-udtryk, der samlet kun medtager værdien for produkter, hvis parameteren date ligger i tidsintervallet fra *AvailableFrom* til *AvailableTo*.
* Indsæt nedenstående testmetode i jeres testprojekt:

[TestMethod]

public void GetValueOfProductsByDate\_Test()

{

Assert.AreEqual(0.0, order.GetValueOfProducts(new DateTime(2018,2,28)));

Assert.AreEqual(10.0, order.GetValueOfProducts(new DateTime(2018, 3, 2)));

Assert.AreEqual(25.0, order.GetValueOfProducts(new DateTime(2018, 3, 3)));

Assert.AreEqual(45.0, order.GetValueOfProducts(new DateTime(2018, 3, 4)));

}

* Kør testen, og tilret din kode (ikke testen) om nødvendigt.

## Øvelse 2.3: Anvendelse af Enumerable.OrderBy()

Udfør følgende:

* Implementér nu en ny metode *SortProductOrderByAvailableTo()* i Order-klassen, som sorterer alle ordrens produkter i stigende orden efter datoen i *AvailableTo*-property. I den nye metode, benyt *Enumerable.OrderBy()* og et passende lambda-udtryk til at udføre sorteringen.
* Indsæt følgende testmetode i testprojektet:

[TestMethod]

public void SortByAvailableToTest()

{

List<Product> result = order.SortProductOrderByAvailableTo();

Assert.AreEqual(3, result.Count);

Assert.AreEqual("Smør", result[0].Name);

Assert.AreEqual("Mælk", result[1].Name);

Assert.AreEqual("Pålæg", result[2].Name);

}

* Kør testen, og tilret om nødvendigt.

## Øvelse 2.4: Mere Sum()

Hvordan kan du helt undlade metoden *Where()* i svaret til øvelse 2.2 ved i stedet at anvende et passende lambda-udtryk i kaldet til *Sum()* (vink: [C# ternary conditional operator](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/conditional-operator))?

* Tilret *GetValueOfProducts(DateTime date)* fra øvelse 2.2, så løsningen kun anvender *Sum()* og ikke *Where()*.
* Kør testen, og tilret om nødvendigt.

## Øvelse 2.5: GetTotalPrice()

På samme måde som for *GetValueOfProducts* skal du også tilrette (overloade) både *GetBonus()* og *GetTotalPrice()*, så der tages højde for blandt andet en angivet dato, hvor henholdsvis bonus og total pris skal beregnes for.

Udfør følgende:

* Overload både *GetBonus()* og *GetTotalPrice()* med metoderne *GetBonus(DateTime date, <delegate-type> <parameter-navn>)* og *GetTotalPrice(DateTime date, <delegate-type> <parameter-navn>)*, hvor man kan angive både dato og bonusberegningen som parametre.
* Indsæt følgende testmetode (bonusberegningen er på 20%):

[TestMethod]

public void GetTotalPriceByDate\_Test()

{

Assert.AreEqual(0.0, order.GetTotalPrice(new DateTime(2018, 2, 28), <bonus>));

Assert.AreEqual(8.0, order.GetTotalPrice(new DateTime(2018, 3, 2), <bonus>));

Assert.AreEqual(20.0, order.GetTotalPrice(new DateTime(2018, 3, 3), <bonus>));

Assert.AreEqual(36.0, order.GetTotalPrice(new DateTime(2018, 3, 4), <bonus>));

}

* Erstat <bonus> med et lambda-udtryk, der beregner 20% bonus.
* Kør testen, og tilret om nødvendigt.

## Øvelse 2.6: Implementér SortProductOrderBy()

I øvelse 2.3 implementerede du metoden *SortProductOrderByAvailableTo()* i Order-klassen. Metoden kan kun sortere ud fra Order-property’en *AvailableTo*. Da Order-klassen indeholder 4 properties, kræver det 4 sorteringsfunktioner for at kunne sortere ud fra en vilkårlig property. I stedet for dette, vil du nu gerne implementere én generel metode *SortProductOrderBy()*, hvor du som parameter kan angive den property, du vil sortere efter. F.eks.

List<Product> result = order.SortProductOrderBy(x => x.Name);

List<Product> result = order.SortProductOrderBy(x => x.Value);

List<Product> result = order.SortProductOrderBy(x => x.AvailableFrom);

List<Product> result = order.SortProductOrderBy(x => x.AvailableTo);

Udfør følgende:

* Implementér sorteringsmetoden *SortProductOrderBy()* med følgende metode-header:
  + List<Product> SortProductOrderBy(Func<Product,object> keySelector)
* Lav selv testmetoder, der tester de 4 varianter.
  + Hint: Det kan tænkes, at det kunne være smart med et loop, hvor der benyttes Assert.AreEqual