# LinkedList

|  |  |
| --- | --- |
| **Læringsmål** | Du kan:   * **1Pv1**: udviklingsbaseret viden om specifikation af abstrakte datatyper […] * **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:   * [Multistrukturel] Du kan beskrive flere kendetegn ved **C#-datatyper** – med fokus på abstrakte datatyper (ADT) (enkelthægtet liste), generisk type * [Unistrukturel] Du kan huske enkelte kendetegn ved **applikationstype** –klassebibliotek (en: class library) |
| **Din forberedelse** | Programmeringssporet:   * [Abstract data types](https://www.linkedin.com/learning/python-data-structures-linked-lists/abstract-data-types) (video: 2:39) * [Data Structure and Algorithms - Linked List](https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm) * [Linked list explained](https://www.linkedin.com/learning/learning-c-sharp-algorithms/linked-list-explained) (video: 2:13) * [Understanding the benefits of generics](https://www.linkedin.com/learning/c-sharp-interfaces-and-generics/understanding-the-benefits-of-generics) (video: 3:51) * [C# Generics Tutorial: Whats and Whys](https://www.youtube.com/watch?v=gyal6TbgmSU) (video: se til og med 8:16) * [Default values of C# types (C# reference)](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/default-values) * Ekstra: [Data Structures: Linked Lists](https://www.youtube.com/watch?v=njTh_OwMljA) (video: 7:42) – forklaret med (Java) kode |

Denne opgave har fokus på flere områder, både nyt og repetition. Som noget nyt skal du arbejde med ’abstrakte datatyper’ (ADT) ved at implementere din egen List-type, hvor du nu selv impleme­nterer en intern datastruktur til at repræsentere din liste uden at bruge noget af alt det, som .NET tilbyder. Du skal også lave en generiske variant af din List-type, så generelt får du uddybet din forståelse af, hvad lister er og kan.

Du får samtidig repeteret opbygningen af en C#-klasse med dets felter, metoder og properties og styrket din algoritmiske forståelse i arbejdet med din egen datastruktur.

**Bemærk, at den næste opgave Ex37-LinkedList2 bygger videre på øvelserne fra denne opgave.**

# Øvelse 1: Terminologi

Del teamet op i mindre grupper og brug **Ordet rundt** til at reflektere over begreberne ”abstrakt datatype”, ”enkelthægtet liste” og ”generisk datatype”.  
Sørg for, at alle får mulighed for at tale.

*Tidsramme: 20 minutter*

# Øvelse 2: LinkedList

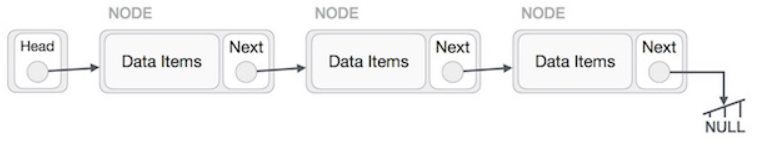
Du skal i denne opgave implementere en lænket liste, et typisk eksempel på en abstrakt datatype. En abstrakt datatype beskriver overordnet, hvilke data en datastruktur kan indeholde, og hvilke operationer der kan udføres på disse data. En datastruktur er en konkret implementering af en abstrakt datatype.

**Benyt parprogrammering.**

## Øvelse 2.1: Algoritmisk forståelse

Du skal implementere en lænket liste, hvor den interne datastruktur realiseres via en **enkelthægtet** liste.

De følgende øvelser er i høj grad træning i algoritmisk forståelse. Så **inden du går i gang** med at kode, så tegn eksempler af lænkede lister på papir, så du får de forskellige variationer med, og forstår hvad der sker, når du indsætter og sletter elementer (Node) fra en enkelthægtet liste:



Dvs. hvad sker der med Head, der peger på første element (Node) i listen, og Next (i Node), som peger på næste element i listen.

Tegn i enkelte trin, hvad der sker med Head og hvert element’s Next-peger, når du henholdsvis:

* indsætter et element først, midt i og sidst i en liste
* sletter et element først, midt i og sidst i en liste

Udfør ovenstående for de fire tilfælde:

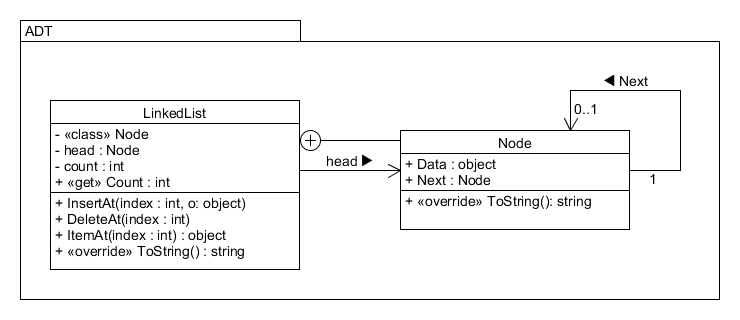
* en tom liste (Head -> null)
* en liste med ét element,
* en liste med to elementer
* en liste med mere end to elementer

Du er nu algoritmisk parat til at implementere LinkedList-klassen.

## Øvelse 2.2: LinkedList-klassen

Læs nøje følgende, og begynd først at kode, når du bliver bedt om det.

LinkedList-klassen er beskrevet neden for i et UML-klassediagram med alle dets felter, properties, constructors og metoder. Studér den nøje. Bemærk den specielle UML-notation  til at angive indlejring af én C#-klasse i en anden C#-klasse (herunder skal Node-klassen defineres internt i LinkedList-klassen).



*Følgende konventioner anvendes:*

* *”private”-attribut startende med lille bogstav er et felt*
* *”public”-attribut startende med stort bogstav er en property (med både get og set)*
* *Operation uden angivet returtype har returtypen ’void’*
* *Stereotypen «class» angiver indlejring af en klasse (se forneden)*

LinkedList-klassen definerer (sammen med Node-klassen) en enkelhægtet lænket liste (singly linked list) og består af en liste af nul, en eller flere elementer (en: nodes). Hvert element i listen defineres af Node-klassen. Det er vigtigt, at den interne datastruktur (den enkelthægtede liste) IKKE må kunne ses uden for klassen, dvs. af brugeren af LinkedList-klassen; overvej kort hvilket OOP-princip er i spil her.

Følgende oplysninger er givet for LinkedList-klassen:

* **- «class» Node**: angiver (sammen med -notationen) at Node-klassen skal defineres internt inde i LinkedList-klassen som en privat (pga. af ’-’) indlejret klasse.
* **- head**: privat felt, der peger på det første element (dvs. første instans af Node-klassen) i den lænkede liste.
* **- count**: privat felt, der angiver det aktuelle antal elementer (nodes) i den lænkede liste.
* **+ «get» Count**: public property, der returnerer det aktuelle antal elementer i listen angivet i det private felt count; kan ikke sættes.
* **+ InsertAt(index : int, o : object)**: indsætter objektet på positionen angivet af index-parameteren; listen er 0-indekseret. Ønsker man at indsætte et element først i listen, angives 0 som index; hvis sidst i listen, angives værdien af Count som index.
* **+ DeleteAt(index : int):** sletter det index’te element i listen (0-indekseret).
* **+ ItemAt(index : int)**: returnerer data (af typen object) for det index’te element i listen (0-indekseret).
* **+ «override» ToString()**: er en override af String()-metoden (fra Object-klassen), der beregner en string-repræsentation af hele listen ved at konkatenere alle ToString()-værdierne for hvert enkelt element i listen. Hvis der er mere end ét element i listen, skal de enkelte tekstrepræsentationer for hvert objekt separeres med ”\n” (ny linje). Se også senere i unit-testen, hvordan det skal se ud.

Der er tre operationer foroven, der benytter sig af en index-parameter. Index (af typen int) refererer til elementets position i listen. Det første element har indeksværdien 0, det sidste element værdien Count-1. Angives et indeks, der er negativt, eller lig med eller større end antal elementer (Count) i listen, skal der kastes en passende Exception (throw new IndexOutOfRangeException(…)). **Dog er det tilladt** at anvende Count som index for InsertAt-operationen for at indsætte et element sidst i listen.

For Node-klassen oplyses følgende:

* **+ Data**: public property, der indeholder elementets data-objekt for elementet i listen. Typen er ”object” og kan derfor være et instantieret objekt af en vilkårlig type i .NET.
* **+ Next**: public property, der peger på det næste element i listen, dvs. en Node-instans. Hvis Next er lig med ”null”, da er elementet det sidste i listen.
* **+ «override» Tostring**(): er en override, der returnerer ToString-værdien af selve data-feltet, dvs. listeelements data-objekt.

Sæt dig godt ind i ovenstående.

## Øvelse 2.3: Klargør løsning

Udfør følgende indledende trin:

* Opret et nyt projekt med navnet ’ADT’ (Abstract Data Types) og projekttypen ’Class Library (.NET Core)’

Hermed har du oprettet et C#-biblioteksprojekt, som ikke kan køres som en selvstændig applikation, men kan anvendes som bibliotek af dine andre C#-applikationer.

### LinkedList-klassen i ADT-biblioteket:

Udfør dernæst:

* Opret en ny klassefil i biblioteksprojektet med navnet ”LinkedList”. LinkedList-klassen i denne fil skal være ’public’.

Indkapsling af Node-klassen i LinkedList-klassen er angivet forneden:

public class LinkedList

{

private class Node

{

public object Data { get; set; }

public Node Next { get; set; }

public override string ToString()

{

return Data.ToString();

}

}

…

}

* Kopier denne fuldt implementerede kode for Node-klassen direkte ind i din LinkedList-klasse som anvist foroven.
* Studér koden (Node-klassen) nøje, så du forstår præcist, hvad den gør.

### Testprojekt

Dit ADT-bibliotek vil under normale forhold blive anvendt af specifikke applikationer, som selv definerer typerne (klasserne) af de objekter, som indsættes i din LinkedList. Du ønsker ikke i denne opgave at implementere en applikation, men vil alligevel gerne teste funktionaliteten af din LinkedList-klasse.

Efter (eller under) din implementering af LinkedList-klassen, skal du gøre brug af en unit-test.

* Tilføj et testprojekt (projekttypen ”MSTest Test Project”) til din løsning.
* Indsæt testklassen angivet i filen ”Ex36-LinkedList.UnitTest.docx” i din unit-test. Du kan finde test-filen i din Git materiale-mappe.

Til testen er der brug for en testklasse betegnet ClubMember. Den skal også indsættes i testprojektet:

* Tilføj en ny klassefil for ClubMember-klassen til testprojektet.
* Anvend følgende definition af ClubMember-klassen (inkl. Gender-typen):

public enum Gender { Male, Female };

public class ClubMember

{

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public Gender Gender { get; set; }

public int Age { get; set; }

public override string ToString()

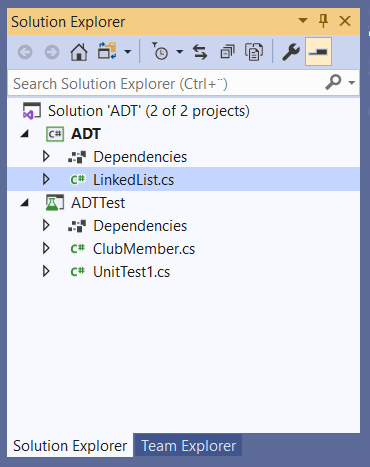
{

return $"{Id}: {FirstName} {LastName} ({Gender}, {Age} years)";

}

}

Din Solution Explorer skulle gerne se nogenlunde ud som forneden:



## Øvelse 2.4: Implementér felter, properties og metoder i LinkedList

Udfør følgende i angivne rækkefølge:

1. Opret felter og properties.
2. Implementér ToString-metoden i LinkedList.
3. Implementér InsertAt()-metoden i LinkedList.
4. Gennemfør de tilsvarende InsertAt-testmetoder.
5. Implementér DeleteAt()-metoden i LinkedList
6. Gennemfør de tilsvarende DeleteAt-testmetoder.
7. Implementér ItemAt()-metoden i LinkedList.
8. Gennemfør de tilsvarende ItemAt-testmetoder.

## Øvelse 2.5: Udvid Linked-klassen med convenience-metoder og -properties

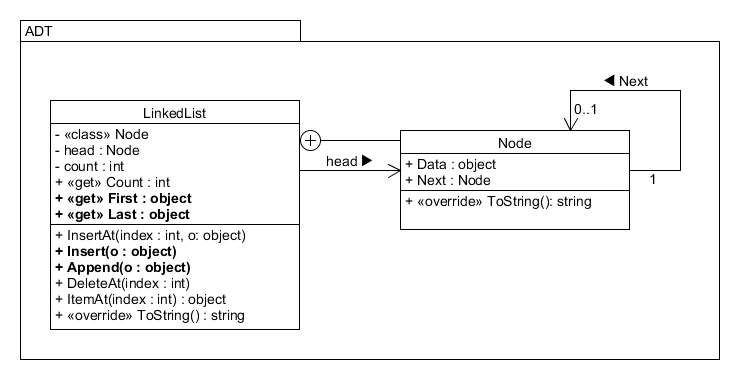
Hvis du ønsker at indsætte et objekt obj først eller sidst i listen list, skal du kalde:

* list.InsertAt(0, obj) henholdsvis list.InsertAt(list.Count, obj)

Dette kan gøres kortere med to convenience-metoder Insert(obj) og Append(obj), hvor du ikke behøver at angive index-parameteren.

Tilsvarende kan det også gøres kortere med to properties First og Last til at returnere det første og sidste element i listen uden at skulle angive en index-parameter i ItemAt()-metoden.

Convenience-udvidelserne er angivet med fed i klassediagrammet forneden:



Udfør følgende:

* Implementér metoderne Insert(object o) og Append(object o) med brug af InsertAt() samt de to properties First og Last med brug af ItemAt() i LinkedList-klassen. Bemærk at First og Last skal returnere værdien null, hvis listen er tom, og ellers det data-objekt, som er henholdsvis først og sidst i listen.
* Fjern udkommenteringen for testmetoden TestMixedElementTypes(), og gennemfør testen.

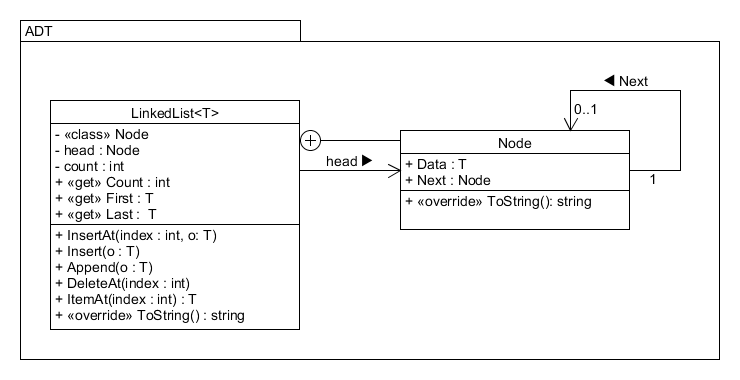
Bemærk at testmetoden TestMixedElementTypes() indsætter forskellige typer objekter (int, string, ClubMember, double) i listen uden problemer.

# Øvelse 3: Implementér den generiske variant LinkedList<T>-klassen

Som nævnt foroven, kan du indsætte objekter hver af vilkårlig datatype ind i en LinkedList uden nogen form for compile-time check, før du kører koden. Dvs. du kan ikke tjekke, om alle objekterne er af den samme type og/eller en bestemt type. Dette skal du rette op på nu ved at implementere en generisk version af din LinkedList-klasse, så du har mulighed for præcist at angive en specifik datatype for objekterne i listen, som kan tjekkes ved compile-time.

## Øvelse 3.1: Implementér den generiske version

UML-klassediagrammet for den generiske LinkedList<T> ser nu ud som følger:



Bemærk at typen ’object’ er erstattet af ’T’.

Udfør følgende:

* Opret en C#-fil til den generiske version med filnavnet ”LinkedListOfT.cs”.
* Kopier nu al koden fra LinkedList-klassen over i filen.
* **Ret klassenavnet fra LinkedList til LinkedList<T>.**
* Gå derefter al kode igennem for alle metoder, properties og felter, så den bliver helt generisk. Med andre ord, introducér T, hvor det er nødvendigt.

Bemærk, at T både kan repræsentere en value-type som int eller double og en reference-type som string eller en objektreference. **Vink:** Hvis du skal returnere en default-værdi for en type T uden at vide, om det er en reference- eller value-type, så kan du bruge nøgleordet **default** (se i forberedelsen).

## Øvelse 3.2: Kør unit-test

Når du er færdig med koden til LinkedList<T>, da udfør følgende:

* Opret en ny unit-test i dit testprojekt.
* Kopier testkode fra testfilen ”Ex36-LinkedListGeneric.UnitTest.docx” fra materiale-mappen ind i denne unit-test.
* Kig alle testmetoder godt igennem, så du forstår, hvad de tester i din kode.
* Kompilér testkoden (Build Solution).

**Bemærk, at det ikke er alle testmetoder, der kan kompileres uden fejl**:

* Identificér fejlen, og overvej/diskutér i gruppen, hvad det netop er, den generiske version sikrer, både compile-time og runtime, som den ikke-generiske LinkedList ikke gør.
* Når I forstår, hvorfor den fejler, udkommentér den ene testmetode, der ikke kan kompileres.

I er nu klar til at køre jeres tests:

* Kør alle generiske tests, og ret i jeres kode om nødvendigt.

## Øvelse 3.3: Test af forskellige typer i LinkedList<T>

Vi har indtil nu primært testet lister med objekter fra ClubMember-klassen. I skal nu også teste med nogle få andre typer:

* Udvid jeres unit test med de 3 testmetoder i ”Ex36-LinkedListGeneric.UnitTest2.docx”.
* Kør denne test og ret evt. fejl.