# Collections i .NET

## Når du kigger i namespacet System.Collections finder du over 10 forskellige klasser. At vælge den rigtige til netop din applikations behov kan derfor godt være lidt svært. I denne artikel vil vi tage et kig på hver enkel collection. Dette skulle gerne hjæ

Skrevet den **03. Feb 2009** af **guidmaster** I kategorien **Programmering / .NET** | ⭐⭐⭐☆☆

Introduktion

En collection er en klasse som giver dig lov til at gruppere andre typer af objekter. Dette kan gøres på mange måde. Eksempelvis i et Dictionary er det nemt at finde et givent objekt igen gennem en nøgle, mens det i en ArrayListe sker gennem et index. Så når du skal til at vælge en måde at gruppere din objekter skal du først tænke over hvad du ønsker at gøre med dine objekter.

En anden ting du skal tage stilling til er om du ønsker at gruppere typestærke eller typesvage collections. Når du vælger at arbejde med typestærke collections betyder det at du kun kan have en bestemt type objekter i en og samme collection.

I denne artikel vil vi bruge en User klasse som vi så vil gruppere på forskellige måder.

```
Public Class User
    Private _UserName As String
    Private _PassWord As String

    Public Sub New(ByVal userName As String, ByVal passWord As String)
        Me._UserName = userName
        Me._PassWord = passWord
    End Sub

    Public Property UserName() As String
        Get
            Return Me._UserName
        End Get
        Set(ByVal Value As String)
            Me._UserName = Value
        End Set
    End Property

    Public Property PassWord() As String
        Get
            Return Me._PassWord
        End Get
        Set(ByVal Value As String)
            Me._PassWord = Value
        End Set
    End Property
End Class
```

**Array klassen**

En af de mest basale og mest udbredte collection typer er Array. Faktisk er det nok lidt af en tilnærmelse at kalde et array for en collection - og array ligger faktisk heller ikke i System.Collection. Et array er typestærkt.

Det vigtigste ved et array er at det har en fast størrelse - der kan altå kun være det antal objekter i det som arrayet er dimensioneret til. Multi array

Følgende kode viser hvordan vi kan lægge 2 User objekter ind i et array:

```
Dim Users(1) As User
Users(0) = New User("mary", "1234")
Users(1) = New User("henrik", "1234")
```

Du kan nu tilgå de enkelte objekter igennem deres index:

```
Users(0).UserName = "Frederik"
```

**ArrayList klassen**

Af navn skulle man tro at ArrayList ville ligne et array. Men det er det ikke. En ArrayListe er ikke typestærk og det er muligt at ændre størrelsen dynamisk.

Følgende kode viser hvordan vi kan lægge 2 User objekter ind i en ArrayListe:

```
Dim Users As ArrayList = New ArrayList
Users.Add(New User("mary", "1234"))
Users.Add(New User("henrik", "1234"))
```

Du skal være opmærksom på at når du initialiserer en ArrayListe vil den som default have plads til 16 objekter. Når man så kommer op over denne grænse vil der ske det at der vil blive oprettet en ny ArrayListe som vil have plads til 32 objekter osv. Derefter vil objekter blive flyttet over i den nye ArrayListe. Derfor: Hvis du ved hvor mange objekter der vil komme ind i din ArrayListe, bør du initialisere din ArrayListe med en størrelse:

```
Dim Users As ArrayList = New ArrayList(20)
```

Ovenstående ArryListe vil have plads til 20 objekter. Hvis der alligevel kommer flere objekter i, vil den blive udvidet til 40 objekter. Overvej derfor hvordan du initialiserer dine ArrayLister.

Da ArrayLister er typesvage er du nødt til at caste objekterne, når du vil arbejde med dem. Hvor vi med Array kunne komme til de properties, der var på objekterne, er vi altså nødt til at gøre følgende, når vi arbejder med ArrayLister:

```
Ctype(Users.Item(0),User).UserName = "Frederik"
```

**Hashtable klassen**

En Hashtable er en typesvag collection bestående af nøgle-værdi par. Dette betyder at hvert objekt du lægger i en Hashtable også skal have en unik nøgle som identificerer objektet. En Hashtable giver dig en mulighed for at få et objekt via dets nøgle.

Følgende kode viser hvordan vi kan lægge 2 User objekter ind i en Hashtable:

```
Dim Users As New Hashtable
Users.Add("m", New User("mary", "1234"))
Users.Add("h", New User("henrik", "1234"))
```

Som du kan se giver man sin Add metode 2 parametere: den første er nøglen og den anden er selve objektet.

Når du skal have et objekt ud af din Hashtable sker dette igennem den nøgle som objektet har. Husk at Hashtable er typesvag, og du er derfor nødt til at caste objekterne:

```
CType(Users("m"), User).UserName = "Mary"
```

Du kan bruge en Hashtable, når du ønsker at arbejde med en collection, som er typesvag, og hvor objekterne kan tilgås via en nøgle og ikke igennem et index.

**Queue klassen**

Du kan sammenligne Queue klassen med den kø der er når du skal betale i dit lokale supermarked. Den der kommer først i køen bliver først ekspederet og derefter er det den næste i køen. Dette kaldes FIFO - First In First Out. Med Queue klassen kan du lave en typesvag collection hvor objekterne bliver ordnet i den rækkefølge de bliver indsat i collectionen.

Følgende kode viser hvordan vi kan lægge 2 User objekter ind i en Queue:

```
Dim Users As New Queue
Users.Enqueue(New User("mary", "1234"))
Users.Enqueue(New User("henrik", "1234"))
```

Vi har tidligere brugt en Add metode til at lægge et objekt ind i vores collection. I en Queue kalder man i stedet Enqueue. Med en Queue collection er det ikke muligt at få fat i et objekt igemmen et index. Det er jo som tidligere nævnt en kø som følger FIFO princippet. Når du vil have fat i det objekt der ligger først i collectionen bruger du metoden Dequeue.

```
Dim firstuser As User = Users.Dequeue
```

Når du har kaldt Dequeue sker der det at man fjerner det første objekt fra collectionen. Det vil sige at næste gang du kalder Dequeue vil du få det objekt der er blevet lagt ind i collectionen som nummer 2.

Du har sikkert bemærket at Queue klassen virker temmelig begrænset - og alligevel er queues noget af det mest brugte i programmering.

### Stack klassen

Hvis du har forstået hvad en Queue collection er, er det også meget simpelt at forstå Stack klassen - det er nemlig det modsatte af en Queue. Hvor en Queue var en FIFO, er en Stack en LIFO - Last In First Out. Ligesom Queue klassen er Stack klassen typesvag.

Følgende kode viser hvordan vi kan lægge 2 User objekter ind i en Stack:

```
Dim Users As New Stack
Users.Push(New User("mary", "1234"))
Users.Push(New User("henrik", "1234"))
```

Metoden du bruger til at lægge et objekt ind i collectionen er Push. Lige som i Queue klassen er det ikke muligt at få fat i et objekt igennem et index. I stedet bruges Pop eller Peek. Forskellen på Pop og Peek er at med Peek får du det sidste objekt i collectionen - med Pop får du også det sidste objekt i collectionen, men desuden bliver objektet fjernet fra collectionen.

```
Dim firstuser As User = Users.Pop
Dim seconduser As User = Users.Peek
```

### SortedList klassen

SortedList er en kobination af et Array og en HashTable. Det vil sige at du kan få fat på et objekt enten gennem et index eller gennem en nøgle. Som navnet på klassen antyder er denne collection sorteret. Collectionen bliver sortereret på den nøgle som objekterne i collectionen bliver tildelt. Du har sikkert regnet det ud, men alligevel: En SortedList er langsommere en en HashTable da den jo skal sorteres hver gang der bliver arbejdet på den

Følgende kode viser hvordan vi kan lægge 2 User objekter ind i en SortedList:

```
Dim Users As New SortedList
Users.Add("m", New User("mary", "1234"))
Users.Add("h", New User("henrik", "1234"))
```

Som tidligere nævnt kan man få fat på objekterne enten gennem et index eller en key. Herunder ser vi hvordan dette ser ud:

```
Dim mary As User = Users("m")
Dim henrik As User = Users.GetByIndex(0)
```

I ovenstående kode brugte vi GetByIndex. Der findes flere insteressante metoder til på en SortedList:

```
GetKey(index) - returnere en nøgle ud fra et index
IndexOfKey(key) - returnere et index ud fra en nøgle
```

### Typestærke collections

De collections klasser vi hidtil har set på har alle været typesvage - undtagen Array. Men hvad med typestærke collections? Hvis du vil arbejde med typestærke collections - eksempelvis en typestærk Users

collection - skal du lave den selv. Det vil sige at for hver klasse du ønsker at lave en typestærk collection må du skrive en speciel collection klasse.

I .NET frameworket finde du 2 abstrakte klasser som gør det nemmere at lave disse typestærke collections: CollectionBase og DictionaryBase. Da denne artikle har til formål at give et overblik over de forskellige colletions der ligger i .NET frameworket, vil vi kun kigge ganske kort på CollectionBase og DictionaryBase.

**CollectionBase**

En meget simpel implementation af en typestærk User collection baseret på CollectionBase:

```
Public Class UserCollection
    Inherits CollectionBase

    Public Sub New()
        MyBase.New()
    End Sub

    Public Sub Add(ByVal user As User)
        MyBase.InnerList.Add(user)
    End Sub

    Default Public Property Item(ByVal index As Integer) As User
        Get
            Return MyBase.InnerList.Item(index)
        End Get
        Set(ByVal Value As User)
            MyBase.InnerList.Item(index) = Value
        End Set
    End Property
End Class
```

Som du kan se nedarver UserCollection fra CollectionBase. Derudover har den en Add metode og en Item som er markeret til at være en Default property. Item propertyen er meget vigtig da det er den der gør at vores UserCollection er typestærk da den kun tillader at der kommer User objekter ind i listen. Derudover returnere Item propertyen også User objekter. Som vi også kan se så bruger vi MyBase.InnerList til at holde på vores objekter. Denne innerlist er faktisk en ArrayList. Vi kan nu bruge vores nye typestærke UserCollection:

```
Dim Users As New UserCollection
Users.Add(New User("mary", "1234"))
Users.Add(New User("henrik", "1234"))
```

Du vil lægge mærke til at Visual Studio nu har Intellisense da VS nu kender den type der ligger i collectionen. Det betyder så at vi ikke længere bliver nød til at caste vores objekter når vi tager dem ud af collectionen:

```
Users.Item(0).UserName = "Mary"
```

**DictionaryBase**

Som vi så med CollectionBase der er bygget op omkring en ArrayListe, så er DictionaryBase bygge op omkring en HashTable. Her rn meget simpel implementation af en typestærk User collection baseret på DictionaryBase:

```
Public Class UserDictionary
    Inherits DictionaryBase

    Public Sub New()
        MyBase.New()
    End Sub

    Public Sub Add(ByVal key As Object, ByVal user As User)
        MyBase.InnerHashtable.Add(key, user)
    End Sub

    Default Property Item(ByVal key As Object) As User
        Get
            Return MyBase.InnerHashtable.Item(key)
        End Get
        Set(ByVal Value As User)
            MyBase.InnerHashtable.Item(key) = Value
        End Set
    End Property
End Class
```

For at bruge denne typestærke UserCollection:

```
Dim Users As New UserDictionary
Users.Add("m", New User("mary", "1234"))
Users.Add("h", New User("henrik", "1234"))

Users.Item("m").UserName = "Mary"
```

**System.Collections.Specialized**

Udover de collections der ligger i System.Collections så findes der også nogle lidt mere eksotiske collections som du finder i System.Collections.Specialized.

ListDictionary Class
HybridDictionary Class
NameValueCollection Class
NameObjectCollectionBase Class
StringCollection Class
StringDictionary Class
CollectionsUtil Class

**Tips og tricks**

Til slut skal vi lige se på hvordan man kan løbe igennem en collection.

For Each

Da mange af de forskellige collections implementerer Ienumerable er det nemt at løbe en collection igemmen med et For Each statement:

```
Dim item As User
For Each item In Users
    Console.WriteLine("Username: " & item.UserName)
Next
```

I VB.NET 2003 er kan det også gøre på følgende måde:

```
For Each item As User In Users
    Console.WriteLine("Username: " & item.UserName)
Next
```

For Each på HashTabeller og implementationer af DictionaryBase

Når du arbejder med en HashTabel eller en implementation af DictionaryBase skal det gøres på en lidt anden måde:

```
Dim item As DictionaryEntry
For Each item In Users
    Console.WriteLine("Username: " & CType(item.Value, User).UserName)
Next
```

**Afslutnig**

Vi har i denne artikel kigget på nogle af de collection klasser der ligger i System.Collections og set hvordan man arbejder med dem. Desuden har vi set på hvordan man kan lave sine egne typestærke collections. Du kan finde mere information om dette emne oppe i Links boksen øverst på siden .
Jeg håber du har fået et lille indblik i hvor stærkt collections er implementeret i .NET frameworket samt at du måske har fået et overblik over de muligheder der ligger i de forskellige typer af collections.

**Kommentar af basementjack d. 05. Nov 2004 | 1**

En jævn god artikel.. Nogle vil nok synes at eksemplerne skulle stå i C# i stedet for VB... Men ikke mig. :)

**Kommentar af md_craig d. 03. Apr 2005 | 2**

Lidt småfejl, ikke noget kriminelt dog.
Desuden syntes jeg artiklen er lidt svag..

**Kommentar af cablenet.dk d. 30. May 2007 | 3**

Fin artikel!

**Kommentar af capn d. 23. Mar 2006 | 4**