Hur fungerar List<T>?

Den generiska[1] klassen `List` används på samma sätt som en vanlig array. Till skillnad mot en vanlig array så måste inte antalet element som `List`-objektet ska lagra vara känt då objektet skapas. Objekt instansierade från `List`-klassen är helt enkelt en array vars storlek kan ökas, eller minskas, om så krävs.

Då du skapar `List`-objekt bestämmer du inte hur många element objektet ska innehålla. Koden som följer skapar ett `List`-objekt som kan innehålla referenser till `string`-objekt. `T` i `List<T>` ersätts med den typ som `List`-objektet ska innehålla.

```
List<string> myList = new List<string>();
```

För att lägga till objekt till `List`-objektet måste du använda metoden `Add`. Koden

```
myList.Add("ett");
myList.Add("två");
myList.Add("tre");
myList.Add("fyra");
```

lägger till fyra strängar till `List`-objektet.

När det väl finns element i `List`-objektet kan du hantera det på samma sätt som en vanlig array. Du kommer åt ett elements värde med hjälp av index:

```
string myString = myList[2]; // tilldelar myString strängen "tre"
```

Du kan använda index för att modifiera ett befintligt elements värde:

```
myList[0] = "fem"; // tilldelar det första elementet strängen "fem"
```

Ett element kan tas bort med

```
myList.RemoveAt(2); // tar bort elementet som innehåller strängen "tre"
```

vilket medför att antalet element i List-objektet minskar med ett.

Med hjälp av en ”for”-sats och egenskapen `Count` kan du stega igenom ett `List`-objekt:

```
for (int i = 0; i < myList.Count; ++i)
{
    Console.WriteLine(myList[i]);
}
```

Satserna ovan skriver ut alla element i `List`-objektet, från det första till och med det sista, med hjälp av index. Använder du ”foreach”-satsen behöver du inte använda dig av index. ”for”-satsen kan istället skrivas med en ”foreach”-sats enligt:

```
foreach (string myString in myList)
{
    Console.WriteLine(myString);
```

[1] List<T> är en generisk klass. Det kan du se på <T>, där T kan ersättas med vilken typ som helst, t.ex. int, string eller en egen typ du skapat.

```
}
```

”foreach”-satsen deklarerar en iterationsvariabel (`string myString`) som automatiskt tilldelas värdet som varje element i arrayen har.

En kopia av ett `List`-objekt skapar du genom att använda en av klassens tre konstruktorer. Satsen

```
List<string> myCopy = new List<string>(myList);
```

skapar en kopia[2] av innehållet i `List`-objektet som `myList` refererar till.

Ett `List`-objekts innehåll kan sorteras. För att innehållet ska kunna sorteras krävs att typen `List`-objektet innehåller implementerar interfacet `IComparable` (eller `IComparable<T>`). Typer i dotnetramverket som `int` och `string` gör detta och ett `List`-objekt med strängar sorteras enkelt med hjälp av metoden `Sort` enligt:

```
myList.Sort();
```

[2] OBS! Om List-objektet innehåller referenser till objekt, som inte är av typen string, skapas ingen kopia av objekten utan det är bara själva referenserna till objekten som kopieras. Vill du att även objekten i sig ska kopieras måste du utföra något som kallas ”deep copy” vilket ligger utanför denna artikels fokus.