Grundlæggende C# syntaks

Sproget C# er et decideret objektorienteret sprog, og det betyder at kode typisk er placeret i såkaldte klasser. Du kan opfatte en klasse som et sted at placere metoder med kode, som kan afvikles.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program1
{
    class Program
    {
        static void Main(string[] args)
        {

        }
    }
}
```

Namespace

Navnet på det benyttede namespace er afhængig af det navn, du har angivet som navn på applikationen – her Program1

Using

I toppen af filen findes en masse using-direktiver, og de har til formål at gøre det nemmere at benytte medlemmer i et namespace. Således vil

```
using System;
```

gøre det muligt at tilgå klasser i System-namespacet – eksempelvis Console-klassen og en af dens metoder:

```
// Console kan tilgås direkte grundet ”using System;”
Console.WriteLine("Test");
```

Så using skaber altså blot en ’genvej’ til klasser eller andre namespaces.

Main

Efter de mange using-direktiver i en konsol-applikation defineres et namespace kaldet Program1, og herefter en klasse kaldet Program (bemærk brugen af tuborgklammer), og i den en enkelt metode kaldet Main. Det er den metode frameworket vil lede efter, når vores applikation afvikles. Kode placeret i denne metode er altså det første, der afvikles, når applikationen startes.

Metoder

Jo mere du kan opdele din kode i små selvfungerende og selvbeskrivende enheder, jo nemmere er det at udvikle, overskue og vedligeholde den. En af måderne, du kan opdele kode, er at udvikle metoder, som udfører en konkret og simpel opgave. Som næsten alle programmeringssprog kan C# håndtere flere forskellige typer af metoder.

Når du benytter en metode i kode, kaldes det at kalde metoden (eller foretage et metodekald). Nogle metoder udfører blot en samling instruktioner, og andre foretager beregninger og returnerer en værdi. Fælles for alle metoder i C# er, at de skal placeres i enten en klasse (class) eller en struktur (struct).

Man kan eksempelvis i én metode kalde en anden metode, og kompileren vil automatisk skabe kode, som flytter programpointeren fra den kaldende metode til den kaldte metode, og her udføre metodens instruktioner. Når alle instruktioner er udført, eller når programpointeren rammer et af flere konkrete kodeord (eksempelvis return), returnerer programpointeren til linjen efter det oprindelige kald. Her fortsættes afviklingen af eventuelle instruktioner.

```
class Program1
{
    static void Main(string[] args)
    {
        Console.WriteLine("Start Main");
        Metode1();
        Console.WriteLine("Slut Main");
    }

    public static void Metode1()
    {
        Console.WriteLine("Start Metode1");
        Metode2();
        Console.WriteLine("Slut Metode1");
    }

    public static void Metode2()
    {
        Console.WriteLine("Start Metode2");
        Metode3();
        Console.WriteLine("Slut Metode2");
    }

    public static void Metode3()
    {
        Console.WriteLine("Start Metode3");
        Console.WriteLine("Slut Metode3");
    }
}
```

Resultat:

```
Start Main
Start Metode1
Start Metode2
Start Metode3
Slut Metode3
Slut Metode2
Slut Metode1
Slut Main
```