

# C# klingar skönt i .NET

C# ("c-sharp") är Microsofts nya programmeringsspråk för .NET-plattformen. C# tar sin utgångspunkt i C-familjen, men är ett modernt, enkelt, typsäkert och objektorienterat språk, utan de svagheter som finns hos traditionell C eller med den komplexitet som belastar C++.

När Microsoft bestämde sig för att skapa en ny plattform för framtida applikationer, idag känd som .NET-plattformen, insåg man också att man saknade ett modernt programmeringsspråk lämpat för uppgiften.

På initiativ av **Anders Hejlsberg**, under sin Borland-tid känd för att först ha utvecklat Turbo Pascal och sedermera första versionen av Delphi, valde man att bygga ett helt nytt språk, döpt till C#. Sin grundsyntax hämtar C# från C/C++, men på högre nivåer liknar C# mer språk som Java och Delphi (hur skulle det kunna vara annorlunda med Anders vid spakarna?).

## Ett enkelt och strikt språk

C# är ett "litet" språk, utan den komplexitet och de inkonsekvenser som belastar många äldre språk, där olika viljor har dragit åt olika håll, när man har försökt förnya dem. Från pascaltraditionen har C# lånat den **stränga typningen** och saknar därför helt förståelse för den slapphet som kännetecknar C-världen.

## Objektorienterad

C# är ett konsekvent objektorienterat språk, med en struktur som påminner om Java och Delphi, t ex har C# både klasser (class) och gränssnitt (interface), och använder klasser för att implementera gränssnitt.

Till skillnad mot C++, men i likhet med Java och Delphi, tillåter inte C# multipla arv av klasser, en "finess" i C++, som har visat sig skapa mer huvudvärk än glädje.

Konstanter, variabler, fält och metoder (funktioner) måste alltid tillhöra klasser eller structs. De kan vara statiska och direkt användas med sin klass (i stil med globala konstanter, variabler och funktioner) eller vara

instansspecifika, dvs endast användas tillsammans med instanser (objekt) av klassen (objekt skapas dynamiskt). Klasser kan nästlas.

Nästan allt i C# är klasser och objekt, förutom några grunddatatyper för logiska värden (bool), tecken (char), heltal, flyttal och egna uppräknings typer (men även dessa kan vid behov tolkas som klasser).

## Komponenter

Precis som Java, Delphi (och Borlands C++Builder) använder C# komponenter med egenskaper och händelser.

## Assemblies

C#, liksom .NET i allmänhet, har stöd för en decentraliserad programmeringsmodell, med återanvändbara moduler av klasser (**assemblies**). För att undvika namnkrockar används namn-områden (**name spaces**) Med nyckelordet **using** öppnar man upp en namn-område (i stil med Java och Delphi). Headerfiler i stil med C/C++ används inte.

## Glöm minnesläckor

C# har en minneshantering med inbyggd skärphantering (den sköts i själva verket av .NET CLR och är därför gemensam för alla .NET-applikationer).

## HelloWorld

Du hittar en flerspråksversion av "Hello world"-klassikern i programrutan. Lägg märke till

- "using System" som gör att vi kommer åt klasser och metoder i standard assembly-namnområdet System, här Console.WriteLine.
- "namespace HelloWorld" som skyddar våra egna namn mot namnkollisioner
- "class Hello" och dess "Main"-metod – alla metoder/ funktioner i C# måste tillhöra klasser.
- "Greetings" är en array av strängar (standardtypen "String"), men som samtidigt är ett objekt. För att ta reda på antalet element anropar vi dess Length-metod.

## C# standardiseras

Microsoft har lämnat över C# till standardisering (till skillnad mot Java där Sun har monopol på utvecklingen), vilket gör att det är firtt att anpassa C# till andra plattformar.

## MSIL

Precis som Java, producerar inte C#-kompilatorer CPU-specifik maskinkod, utan ett slags "mellankod" som Microsoft döpt till "Microsoft Intermediate Language", **MSIL**. All kod i .NET (inkl assemblies) är kompilerad till MSIL

### Flerspråksversion av HelloWorld i C#

```
using System;

namespace HelloWorld
{
    class Hello
    {
        static void Main()
        { String[] Greetings = {"Hello world!", "Hej värld!",
                               "Hola Mundo", "Hallo Welt",
                               "Salut le Monde", "Ahoj, svet"};
          for (int i = 0; i < Greetings.Length; i++)
            Console.WriteLine(Greetings[i]);
        }
    }
}
```

och kan sedan intepreteras av en .NET "virtuell maskin" eller kompileras till maskinkod av en "just-in-time" (JIT)-kompilator anpassad till önskad plattform (idag för Intel 80x86-familjen och Intel Armstrong för mobila enheter).

Till skillnad från Sun's sk "bytekod", som är skräddarsydd för Java, är dock MSIL inte låst till C#.

Alla programspråk och kompilatorer anpassade till .NET (som Visual

Basic .NET, Visual C++ .NET, Visual J# .NET, Eiffel för .NET och Delphi för .NET) skapar MSIL-kod, vilket det gör möjligt att dela kod och assemblies mellan olika programspråk, CPU:er och grundplattformar (Windows, Pocket PC, etc.).

### **C#-kompilatorer**

En gratis C#-kompilatorer (radkompilator) ingår i .NET Framework

1.1, som kan laddas ner från Microsoft.

I **Visual C# .NET 2003 Standard** får du C# med Microsofts visuella utvecklingsmiljö och i samtliga **Visual Studio .NET 2003**-versioner ingår också C#.

Borlands alternativ är **C#Builder for .NET**, med en utvecklingsmiljö med Borland-känsla och en rad unika tillbehör.

