3.3 Statiska datamedlemmar och metoder

När vårt allra första C# program **MyFirst** introducerades sades så här om metoden **Main()**:

> ”Det är Virtual Machine (VM) som exekverar vårt program **MyFirst** genom att anropa metoden **Main()**. För att detta anrop ska kunna utföras behövs den s.k. *modifieraren* **static** i metodens huvud.”

Sedan dess har vi använt **static** tillsammans med **void** i alla våra program i huvudet till metoden **Main()**:

```
static void Main()
```

En allra första förklaring gavs så här:

static	att den kan anropas utan att skapa objekt av klassen **MyFirst**,
void	att metoden **Main()** inte returnerar något värde.

Men nu när vi har mer kunskap om klasser och objekt ska vi precisera dessa förklaringar. **void** tillhör kategorin *returtyper*. Medan **public** och **private** är s.k. *åtkomstmodifierare*, är **static** en s.k. *allokeringsmodifierare* som vi kommer att behöva när vi skriver egna metoder som ska anropas från **Main()** utan att skapa ett objekt. Men **static** kan inte bara skrivas framför metoder utan även framför klasser och datamedlemmar. Vi tar upp först datamedlemmar som man kan dela upp i två kategorier:

Klass- och instansvariabler

En icke-statisk datamedlem kallas för *instansvariabel*. *Instans* är synonym till objekt. Följande klass visar ett exempel på båda dessa kategorier:

```
// StatDemo.cs
// Klass med två datamedlemmar, en statisk och en icke-
// statisk

class StatDemo
{
   public static int klassVar; // Klassvariabel allokeras här
                               // i klassen.
                               // Behöver inget objekt.

   public int instVar;         // Instansvariabel kommer att
                               // allokeras i varje objekt
                               // som skapas av denna klass
}
```

Programmet nedan demonstrerar skillnaden mellan klass- och instansvariabler i en loop:

```
// StatDemoTest.cs
using System;

class StatDemoTest
{
   static void Main()
   {
      int i = 0;
      Console.WriteLine("\nKlassvariabeln skapas och initi" +
         "eras i klassen till: " + StatDemo.klassVar + '\n');
      do
      {
         StatDemo obj = new StatDemo();      // Nytt objekt i
                                             // varje varv
         Console.WriteLine(
           "Samma klassvariabel ökar i varje varv:\t\t    " +
                                  StatDemo.klassVar + '\n'  +
           "Ny instansvariabel skapas i varje objekt:     " +
                                     obj.instVar + '\n');
         StatDemo.klassVar++; // Ökar löpande
         obj.instVar++;       // Ökar i varje objekt från 0-1
      } while (i++ < 4);      // Fem varv
   }
}
```

Programmet ovan producerar följande resultat när det exekveras:

```
Klassvariabeln skapas och initieras i klassen till: 0

Samma klassvariabel ökar i varje varv:              0
Ny instansvariabel skapas i varje objekt:     0

Samma klassvariabel ökar i varje varv:              1
Ny instansvariabel skapas i varje objekt:     0

Samma klassvariabel ökar i varje varv:              2
Ny instansvariabel skapas i varje objekt:     0

Samma klassvariabel ökar i varje varv:              3
Ny instansvariabel skapas i varje objekt:     0

Samma klassvariabel ökar i varje varv:              4
Ny instansvariabel skapas i varje objekt:     0
```

På första raden skrivs ut klassvariabeln **`klassVar`**:s initialvärde **`0`** genom att i programmets första utskriftssats före loopen referera till **`StatDemo.klassVar`** dvs

klassen **StatDemo**:s statiska datamedlem **klassVar**. Modifieraren **static** framför deklarationen av **klassVar** i klassen **StatDemo** gör att vi kan referera till denna variabel med klassnamnet utan att behöva att skapa ett objekt.

I utskriften ovan visas klassen **StatDemo**:s både statiska och icke-statiska datamedlemmar. Man ser att klassvariabelns värde löpande ökar, medan instansvariabeln visar värdet **0**. Utskrifterna är resultat av en loop i programmet som har fem varv motsvarande räknaren **i**:s fem värden **0-4**. I varje varv skapas ett objekt av typ **StatDemo**. Både objektets (dvs instansens – instans är bara ett annat ord för objekt) och klassens variabel ökar sina värden med **1** i varje varv av loopen. Detta sker i satserna **StatDemo.klassVar++;** och **obj. instVar++;** som står sist i loopen. Klassvariabeln kommer ihåg sina gamla värden från loopens gångna varv eftersom dess ökning sker i en och samma minnescell genom hela programmet och därför löpande. Instansvariablerna däremot skapas i varje varv av loopen i sina objekt på nytt, initieras till **0** och skrivs ut, ökar sedan till **1**, men ”dör” i början av nästa varv när ett nytt objekt skapas. Närmare bestämt, överskrivs deras adress i referensen **obj** med det nya objektets adress. Därför får de bara värdena **0** och **1** tilldelade av vilka endast **0** skrivs ut eftersom utskriftssatsen står före ökningen. Det handlar om olika objekt med olika instansvariabler i varje varv lagrade vid olika adresser.

I programmet ovan behandlades **static** för datamedlemmar. Nu ska vi undersöka **static** för metoder. Låt oss först sammanfatta vad vi vet om **static**.