Oscar Antbring

PROGRAMMERING 1

DOCENDO

Våra böcker och tillhörande produkter är noggrant kontrollerade, men det är ändå möjligt att fel kan förekomma. Vi tar gärna emot förbättringsförslag.

Tryckt av Elanders
Utgiven 2024

ISBN: 978-91-7531-165-4
Artikelnummer: 3116

Författare: Oscar Antbring

Omslag: Docendo AB
Bild på omslaget: © Adobe Stock

Innehållsförteckning

2 Programutveckling i .NET

I detta kapitel ska du få läsa om **.NET** som är ett av de absolut mest populära ramverken för att utveckla program. Det är ett oerhört mångsidigt ramverk som innebär att du kan utveckla allt från mobila appar, webbappar, spel och vanliga skrivbordsprogram för Windows.

Ett ramverk är omfattande och innehåller många saker. Den mest intressanta delen för dig just nu är biblioteket av färdig kod som du kan använda. Detta är något som du kommer jobba med redan när du skriver dina första koder, men det är nog först en bit in i läroboken och i din kodning som du är införstådd med det och hur dessa resurser kan användas.

En annan viktig del i ramverket är en tjänst som ser till att din kod kan bli ett körbart program. Detta innebär dels att vi *kompilerar* koden vilket betyder att koden du skrivit görs om till ett språk som datorn förstår, dels *exekverar* koden som innebär att programmet körs.

Tidigare var **.NET** strikt begränsat till Microsofts produkter. Dessutom var Microsoft känt som en motståndare till öppen källkod, vilket sågs som ett hot mot företaget. Öppen källkod innebär att all den kod som ett program bygger på är helt fri för alla användare att ta del av, och du får också lov att ändra i koden och dessutom publicera det modifierade programmet. Ett av de mest kända programmen som bygger på öppen källkod är ett operativsystem som heter Linux. Och eftersom Microsoft Windows också är ett operativsystem så kan du tänka dig Microsofts inställning mot Linux.

Det går att hoppa över detta kapitel tills vidare och det ska inte inverka på din möjlighet att hänga med i de inledande kapitlen i läroboken (och övningarna i den tillhörande övningsboken). Däremot kommer du vilja att förstå mer om hur delar av .NET fungerar när koderna blir mer komplexa, och när du mer aktivt börjar jobba med resurser som ramverket tillhandahåller.

I nästa kapitel så skriver vi om hur du påbörjar alla koder som så kallade konsolprojekt, vilket vi berättar mer om då. I och med att du skapar ett sådant projekt så kommer en del färdig kod att skapas, och en del i att få en insikt i ramverket är att förstå denna kod, eller *kodskal för ett konsolprojekt* som vi kommer att kalla det för.

Har du förresten någon gång sökt på arbetsförmedlingen efter utlysta tjänster inom programmering? För att se vilka kompetenser som är mest eftertraktade. Då har du säkert noterat att det är vanligt att företag söker efter **.NET**-utvecklare. Sedan brukar tjänsten specificera vilket språk som du ska jobba med och vilken typ av applikationer som du ska bygga. För en sak som du säkert redan förstått är att ramverket är stort, eller snarare ramverken eftersom det finns flera varianter av det.

Ska vi beskriva **.NET** med ett ord (eller ett par i alla fall) så landar formuleringen i att det är en *plattformsoberoende utvecklarplattform*, vilket råkar vara den beskrivning som Microsoft använder sig av. Ett annat begrepp som ibland förekommer när Microsoft slår på stort är ekosystem.

Ramverket har inte alltid varit plattformsoberoende. Den första versionen av ramverket släpptes för ungefär tjugo år sedan och då hette ramverket **.NET Framework**. Det växte och svällde, lappades och lagades samtidigt som tekniken i omvärlden utvecklades i rasande fart. För er som upplevt det tidiga 2000-talet så kan ni skriva under på att det var en helt annan tid då. Detta med att exempelvis utveckla appar för mobiltelefoner fanns inte riktigt på kartan, eftersom smartmobiler lanserades först flera år senare. Och sociala medier var knappast en grej ännu, Facebook lanserades visserligen 2004, men det tog flera år innan det slog igenom på allvar.

Innan vi börjar skriva om ramverket vill vi påpeka att flera begrepp dyker upp och som säkerligen upplevs väldigt abstrakta för dig som är ny i programmering (eller i C#). Det finns över huvud taget inga krav på att du ska förstå begreppen redan nu. Eller vi kan säga så här, du *kommer* inte förstå begreppen fullt ut förrän du kommit längre i din programmering. Det enda rimliga egentligen är att du kan förväntas förstå begreppen först när du jobbar med *objektorienterad programmering* i slutet av denna lärobok.

Men vi ska i alla fall så ett litet frö som får kultiveras genom boken och som sedan får blomma ut när du skriver större och svårare koder.

Däremot är det viktigt att du redan från start förstår den kod som *du* skriver, och egen kod kommer du börja skriva i kapitel 4.

Öppen källkod

Någon gång under 2015 så tänkte sig nog Microsoft att det inte riktigt var hållbart att fortsätta underhålla **.NET framework**. I och med alla lappningar och lagningar samt utbyggnader, så hade ramverket blivit väldigt svullet och långsamt. En annan svaghet som ramverket hade var att en utvecklare enbart kunde skriva program för Windows. Öppen källkod hade också växt i popularitet, men traditionellt såg alltså Microsoft öppen källkod länge som ett hot. Och att utveckla mjukvara med öppen källkod var där och då otänkbart.

> Öppen källkod innebär att koden som ett program bygger på är helt öppen och att det är fritt fram för vem som helst att ändra och bygga ut koden efter egna behov. Vissa ändringar från privatpersonen och andra företag blir också officiella förändringar i koden.

Microsoft och .NET Core

Men tiderna förändrades och likaså Microsofts filosofi. Idag är Microsoft en av världens största bidragsgivare av program med öppen källkod. I och med ett nytt och mer öppet förhållningssätt ville företaget också att program som utvecklades inom ramen för **.NET** inte längre skulle vara begränsade till Windows.

Man kan säga att Microsoft rättade in sig i ledet med **.NET Core**.

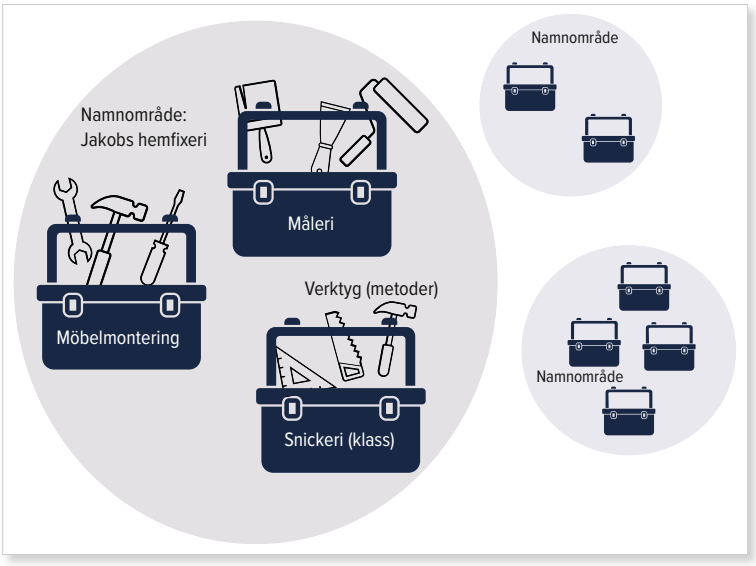
Där någonstans landade Microsoft och släppte uppföljaren som kom att lösa de problem som det äldre ramverket drogs med. En fördel med det nya ramverket var att det var betydligt snabbare och lättare än sin föregångare, samtidigt som det också var med modulärt. Att ramverket är *modulärt* innebär att du får mindre färdig kod i grundutförandet att arbeta med, men att utvecklaren i stället kan välja att importera bibliotek som behövs. I det tidigare ramverket fanns *alla* resurser med från start, och det var därmed många resurser som inte användes. Detta var också en anledning till att det blev långsamt, då programmet fick släpa på mycket resurser i onödan.

I läroboken så har vi gjort en liknelse med en fixare som har många olika verktygslådor för olika typer av jobb. Om fixaren får åka på ett jobb att måla om ett hus så tar han eller hon inte med sig verktygslådan med verktyg för att snickra en altan eller meka med en bil, utan bara den verktygslåda med verktyg som är användbara för uppdraget. Detta är ett *modulärt* förhållningssätt.

Klasser och metoder

För att ytterligare förstå vad motsvarande verktyg kan innebära för dig som programmerare så behöver du få en viss insikt i vad *klasser* och *metoder* är. Du kommer faktiskt att jobba med sådana redan i dina första program även om du troligtvis inte kommer att vara medveten om det. Framåt slutet av boken så ska du skapa egna klasser och metoder. Då kommer du att ha betydligt bättre förutsättningar att förstå allt detta.

I bilden kan du se verktygslådor som används för olika ändamål. Om du öppnar verktygslådan som du använder för att montera en möbel så förväntar du dig att det finns verktyg som är användbara för det arbetet, och inte en målarpensel eller en rörmokartång.



Vi har ju pratat om modularitet: Om fixaren åker för att måla om ett hus så tar han eller hon ju inte med sig alla verktygslådor, bara den som används vid målning.

Verktygslådan i exemplet motsvarar en klass. Precis som att en verktygslåda har ett särskilt ändamål, så är det också så med klasser. Och det finns många - med betoning på *många*.

Tack vare ramverket **.NET** så kan vi använda alla dessa klasser och att de är organiserade på ett sätt så att vi inte blir överväldigade. Eller att vi blir mindre överväldigade i alla fall. Microsoft har nämligen ordnat det så fint att klasserna är organiserade i så kallade *namnområden* (engelska **namespace**). I ett namnområde så samlas klasser som används för samma ändamål. Alla namnområden utgår ifrån namnområdet **System** som är det mest övergripande.

3 Visual Studio och konsolprojekt

Välkommen till detta kapitel som har fokus på utvecklingsmiljön **Visual Studio**.

Du kommer att få läsa om värdefulla verktyg och funktioner i programmet som kommer underlätta din vardag som programutvecklare.

I samband med att vi snart kommer berätta mer om konsolprojekt så är det oundvikligt att inte nämna centrala begrepp som *klass, namnområde* och *metod*. Detta är begrepp som här och nu säkerligen kommer att upplevas abstrakta. De går inte riktigt att ta på innan du börjat skriva lite mer komplexa koder framåt slutet av läroboken.

Men vi gör vårt bästa i att berätta om **Visual Studio** utan att du ska hamna på för djupt vatten. Precis som i föregående kapitel är det inget krav eller ens rimligt att du förstår begreppen redan nu. Det handlar mer om att du ska få en idé om vad det handlar om och samtidigt sår vi ett frö som får kultiveras under din resa. I slutet av kursen kommer du ha en annan förståelse för dessa begrepp, var så säker.

```
              10 File(s)      7 898 931 bytes
               9 Dir(s)  813 095 399 424 bytes free

C:\Program Files\Notepad++>cd themes

C:\Program Files\Notepad++\themes>dir
 Volume in drive C is Windows
 Volume Serial Number is 840D-22C0

 Directory of C:\Program Files\Notepad++\themes

2023-07-09  14:40    <DIR>          .
2023-07-09  14:40    <DIR>          ..
2022-12-16  17:43            94 338 Bespin.xml
2022-12-16  17:43            92 990 Black board.xml
2022-12-16  17:43            93 072 Choco.xml
2022-12-16  17:43           111 100 DansLeRuSH-Dark.xml
2023-04-02  18:58           183 373 DarkModeDefault.xml
```

Utöver att lära dig viktiga verktyg i utvecklingsmiljön så ska du också kunna skapa ett konsolprojekt. En konsol är i praktiken ett gränssnitt för att en användare ska kunna kommunicera med en dator. Och traditionellt syftar detta till en fysisk enhet som ett tangentbord och en skärm, idag syftar man främst på en mjukvara som är ett fönster utan grafik där du kan skriva olika kommandon. Ett *kommandorads-gränssnitt*, eller mer vanligt en *kommandotolk*.

Om du använder **Microsoft Windows** kan du starta denna kommandotolk genom att skriva kommandot **cmd** i startmenyn. På **Mac OS** hittar du terminalen under verktygen. I bilden på föregående sida kan du se kommandotolken från Windows och där du exempelvis kan skriva kommandon för att navigera mellan olika kataloger och lista filerna i dessa.

Konsolprojekt är den typen av projekt som du uteslutande kommer att jobba med genom boken och från och med nästa kapitel då du ska börja skriva dina egna koder.

Men först ska vi alltså berätta lite om hur du kommer i gång med **Visual Studio**.

Installera Visual Studio

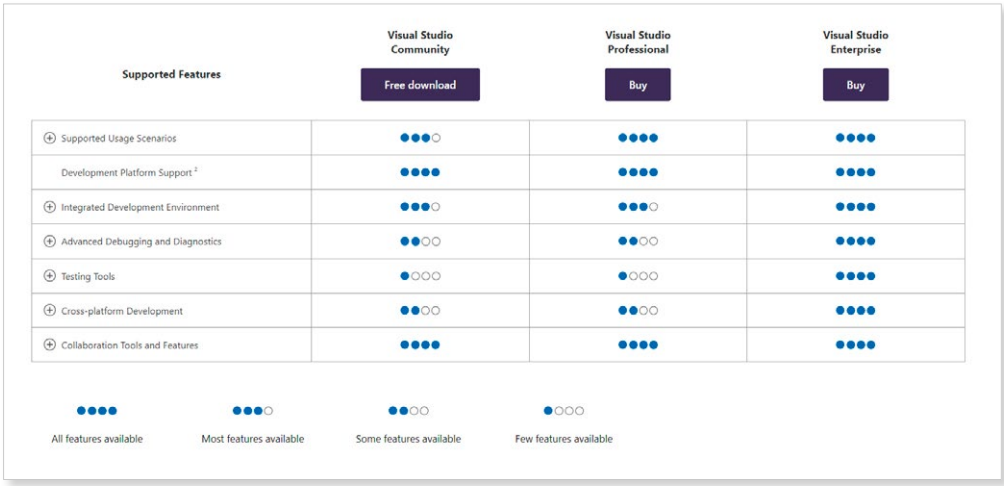
Innan du kan börja skriva dina första koder krävs något program där du kan göra det, att skriva dina koder i alltså. Utöver detta vill vi också kunna göra om koden till ett fungerande program, alltså att koden kan kompileras till maskinkod som datorn kan exekvera (som vi skrev om i förra kapitlet).

Då finns alltså **Visual Studio** som kan göra allt detta och som dessutom är gratis och lätt att komma i gång med. Sedan kommer du snart att märka att det finns många hjälpmedel som verkligen kan underlätta i utvecklingsarbetet. Du kommer att läsa om sådana i detta kapitel, men du kommer lära dig ytterligare sådana längre fram i boken. Och det är ju egentligen först när du verkligen kommit igång att koda som du på riktigt kan förstå hur dessa verktyg kan hjälpa dig i utvecklingsarbetet.

Du kan känna till att utvecklingsmiljön finns i tre olika versioner: **Visual Studio Community**, **Professional** och **Enterprise**. Den version som du ska använda är den förstnämnda och det är den som åsyftas när vi skriver **Visual Studio**. De övriga två versionerna är inte gratis men har inte heller oväntat fler finesser och verktyg, vilka dock inte är några som du behöver för kursen du läser just nu. Dessa versioner är mer riktade mot yrkeslivet, vilket ju också framgår av deras namn.

Det engelska begreppet för en komplett utvecklingsmiljö är **integrated development environment** eller **IDE**. Man kan säga att en *integrerad utvecklingsmiljö* har i syfte att vara ett komplett program för programmeraren. Detta innefattar i stora drag en kodeditor, flera funktioner som stöder utvecklingsarbetet samt verktyg för att felsöka och därefter kompilera koden.

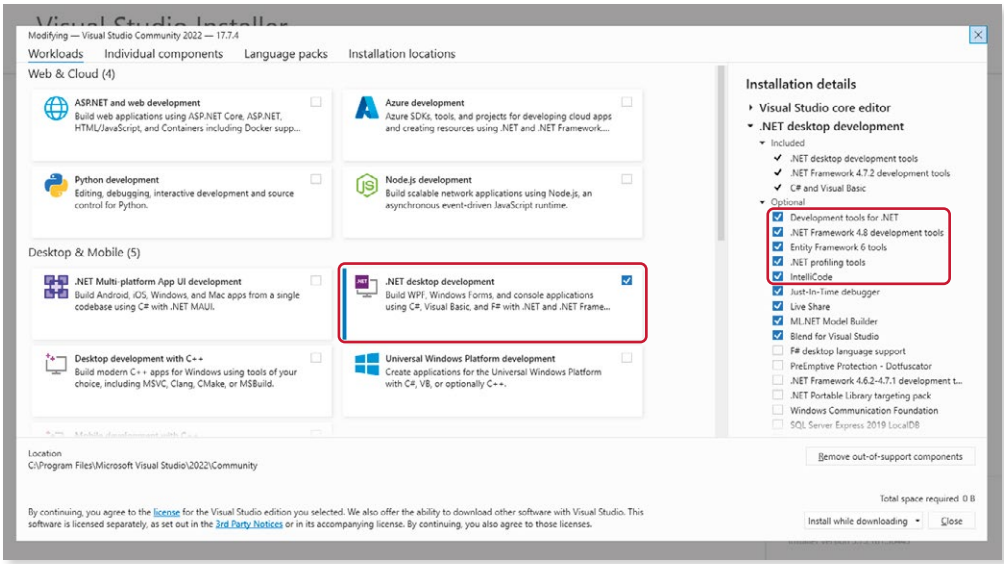
> På webbsidan https://visualstudio.microsoft.com/vs/compare/ kan du se en jämförelse mellan de olika versionerna av Visual Studio.



Även om många av dessa finesser är främmande för oss så kan du i alla fall få insikten att det finns flera sådana som på olika sätt underlättar utvecklingsarbetet. Vi kommer snart titta på funktioner som är relevanta för oss i arbetet med denna kurs.

Men om du klickar på ikonerna med plus-tecknet så kommer du se verktyg som används för att avlusa och diagnostisera koden, och verktyg för att exempelvis utveckla program för flera plattformar.

Att ladda hem och installera **Visual Studio Community** ska inte innebära några större bekymmer. Däremot ska du se till att du har markerat **.NET desktop development** enligt bilden nedan. Denna **workload** (samling av komponenter) innebär att du kan utveckla konsolprojekt som är den projektform vi ska jobba med.



4 Hello World

Ett särskilt varmt välkommet önskar vi er till detta kapitel. Stunden är kommen för att du ska börja skriva dina första körbara koder. Sedan kommer du bara att utveckla dina kunskaper genom kommande kapitel och dina program kommer bli alltmer omfattande och användbara. Och roliga att skriva koden för.

Inledningsvis så kommer du säkerligen bli varse hur petigt det är med programmering. Ett felaktigt tecken gör koden okörbar. Lyckligtvis jobbar du i en utvecklingsmiljö som **Visual Studio** som hjälper dig att skriva en korrekt kod. Detta skrev vi om i föregående kapitel.

Detta kapitel kommer att bli ett förhållandevis kort sådant och utgör startskottet på din resa i kodandets värld. Du ska skriva ett program som skriver ut en klassisk text ”Hello World” som brukar utgöra det allra första programmet som en programmerare skriver.

Ja, utskriften ”Hello World” har en liten särskild plats i många programmerares hjärtan. Den brukar alltså utgöra ett grundläggande program som inte tar emot någon text alls från tangentbordet, utan enbart skriver ut en text. Programmet bygger därmed på en kod som innefattar den allra mest grundläggande syntax i ett programmeringsspråk.

Dessutom brukar programmeraren vilja testa utvecklings- och körmiljön så att koden kan kompileras korrekt och bli ett körbart program, och detta ska inte underskattas. För att du ska kunna köra mer komplexa program längre fram så är det ju rimligt att ditt ramverk kan hantera en enkel utskrift. Vi kan kalla detta för en hälsocheck av utvecklingsmiljön.

Traditionen att skriva ut texten ”Hello World” på skärmen har gedigna historiska anor. Vanligtvis brukar man hänvisa till boken *Tutorial Introduction to the Language B* från 1972 och att frasen användes för kodexempel i den. På denna tid var det också en utmaning att hantera längre fraser, så det var alltså en större prestation att lyckas skriva ut ”Hello World” korrekt, jämfört med exempelvis ”Hi”.

Kodsyntax betyder de regler för hur kod skrivs i ett särskilt språk. Hur olika tecken används och vilka instruktioner som kan användas.

En spännande aspekt i detta är hur alla programmeringsspråk kan åstadkomma denna utskrift, men att kodsyntax *för* detta ändamål skiljer sig. Vi ska inte förmedla någon syntax från något annat språk än C# i denna bok men det finns massor av resurser på webben som både skriver mer om frasen ”Hello World” och som dessutom visar kodsyntax från olika språk för att åstadkomma utskriften.

Programsatser och uttryck

Genom boken kommer du att skriva många *programsatser* (engelska **statements**).

> En programsats är en åtgärd i koden som avslutas med ett semikolon. Och som kan liknas vid en mening i vanligt skriftspråk.

När vi syftar till en programsats så handlar det om en kod som avslutas med ett semikolon. Och koden gör *en* sak. Längre fram i boken så kommer du börja jobba med villkor och då kan man också prata om *uttryck* (engelska **expressions**).

I kodexempel 4:1 så har vi skrivit en programsats på rad 9 som skriver ut texten ”Hello World” på skärmen.

Vi kommer så långt som möjligt använda begreppet programsats i boken. Det finns många definitioner på en programsats och vi kan exempelvis kalla den för en enskild enhet för exekvering som inte utvärderar någonting. Vi har nämnt att en annan typ av kod i stället *kan* utvärdera något och då kallas det för uttryck.

Om du befinner dig i ditt hus och du bestämmer dig för att gå ut oavsett väder så är det en sats. Men om du i stället funderar över att gå ut, men bara om temperaturen är över 20 grader, så är det i stället ett uttryck. Innan du går ut kommer du först utvärdera om vädret är tillräckligt bra.

Kodskalet för ett konsolprojekt

Efter att du läst kapitlet om Visual Studio så har du lärt dig hur du skapar ditt första konsolprojekt. Likaså påpekade vi att det finns ett val att göra i samband med att du skapar ett projekt. Alternativet **Do not use top-level statements** ska vara markerat om du vill ha det kompletta kodskalet som du kan se i kodexempel 4:1.

Alternativet är att avmarkera detta och då kommer du enbart se en kod för en utskrift samt en kommentar som länkar till en sida på Microsoft som beskriver detta nya sätt att koda på.

Du får själv välja vad som känns bäst, men framåt slutet av boken så vill vi att du arbetar med det kompletta kodskalet för ett konsolprojekt som du kan se i kodexemplet. Vi vill också passa på att marknadsföra bilaga A där vi dissekerar denna kod, där vi alltså går igenom var och en av kodraderna relativt djupt. Tanken är att du som elev ska kunna ta del av beskrivningarna i denna bilaga när det känns som att tiden är mogen, alternativt att du återkommer till bilagan flera gånger under resans gång.

Utskriften ”Hello World”

På rad 9 i kodexemplet så skriver vi alltså koden för en utskrift. Vi har också lagt till en viktig kodrad på rad 10 som gör att programmet stannar upp och väntar på att användaren trycker på en tangent. Om du inte har med den kodraden så kommer konsolfönstret att öppnas och stängas så fort att du inte hinner blinka. I alla fall fungerar det så i vissa utvecklingsmiljöer.

```
1.  // Kodexempel 4:1
2.  using System;
3.  namespace exempel
4.  {
5.      class Program
6.      {
7.          public static void Main(string[] args)
8.          {
9.              Console.WriteLine(”Hello World!”);
10.             Console.ReadKey(true); //programsats
11.         }   //Här avslutas kodblocket som tillhör metoden Main()
12.     Console.WriteLine(”Hello World!”);
```

På kodrad 11 så har vi skrivit en kommentar. Den texten inleds med två snedstreck ”//”. Om C# stöter på denna uppsättning tecken så kommer den raden i koden att ignoreras vid kompilering.

Om du vill prova att köra en kod utan en särskild kodrad så tipsar vi dig om att inte ta bort kodraden utan enbart göra om det till en kommentar. Det gör du alltså genom att placera två snedstreck först på kodraden vilket innebär att efterföljande text på raden blir en kommentar.

Fördelen då är att du kan prova att köra koden utan just den kodsatsen men att det är enkelt att återigen aktivera kodsatsen genom att ta bort snedstrecken.

```
13. //Console.ReadKey(true); Detta är nu en kommentar och kommer ignoreras
```

Vi har också skrivit kommentarar i exempel 4:1. Här skriver vi att ett *kodblock* som tillhör metoden **Main()** avslutas. Ett kodblock utgörs av en eller flera rader kod som befinner sig i en uppsättning klammerparenteser. Kodraderna 9 och 10 tillhör ett kodblock för en metod.

Mer om kommentarer

Du har nu stött på begreppet *kommentarer* som är ett mycket viktigt redskap för programmeraren.

Vi ska egentligen bara förklara syftet med kommentarer på ett överskådligt sätt samt berätta hur du kan kommentera på olika sätt. Men vi vill gärna uppmuntra att du söker ditt sätt att kommentera på. För kommentarer i koden är något som du kommer sätta din personliga prägel på. Här finns inga direkta regler, inga facit. Däremot riktlinjer.

Kommentarer har som främsta syfte att förklara din kod och för andra programmerare. Men du kan också tänka att du ska förklara koden för dig själv. Om du påbörjar en kod och sedan åker på semester till Kanarieöarna i en vecka, så kommer du behöva lägga mycket tid på att sätta dig in i koden när du kommer hem igen, såvida du inte kommenterat koden ordentligt.

Du kommer märka att man glömmer koden och logiken väldigt fort när man lär sig programmering. Det räcker med att du håller uppe ett par dagar så kan det kännas som all din inlärda kunskap och förståelse är som bortblåst.

Det är mindre viktigt, eller egentligen onödigt, att skriva kommentarer som påpekar vad en självklar kodrad gör. Exempelvis att kommentera att du nu ska skapa en variabel som kan lagra ett tal, eller att du skriver en kommentar att nästa kodrad skriver ut en text på skärmen.

Men, och detta är ett väldigt viktigt *men*, dina koder kommer i början bestå av enkla koder, som att skapa variabler och skriva ut texter på skärmen. Vi vill uppmuntra att du skriver kommentarer redan från start. Så till en början kommer dina kommentarer att bli mer av den karaktär som vi beskrev som mindre viktig, eller onödiga.

Men då får de inte betecknas som onödiga i dina första program, tvärtom.

Dessutom är det ofta så att du som elev vill kommentera koden extra mycket för att förmedla till läraren att du faktiskt förstår vad koden gör.

```
1.  //Kodexempel 4:2
2.  /*Kodrad följer för metoden Main() med flera kryptiska ord som du inte behöver
3.  förstå ännu*/
4.  public static void Main(string[] args)
5.  {
6.          //Detta är kodblocket som tillhör metoden Main()
7.          Console.WriteLine(”Hello world”);                    //”Hello world” är ett så kallat argument
8.          Console.Write(”Press any key to continue . . . ”);
9.          Console.ReadKey(true);
10. }
```

PROGRAMMERING 1 MED C#

Boken är skriven för dig som vill lära dig programmering med C# från grunden. Att lära sig programmering handlar inte bara om att lära sig vad man ska skriva för kod för att åstadkomma något, utan också att du ska träna upp din förmåga att se logiska mönster och strukturer. Att kunna vrida och vända på problem och lösningar. Kodförståelse är ett bärande begrepp och utgångspunkten ska vara att du alltid ska förstå den kod du skriver.

I boken får du bland annat lära dig hur du hanterar variabler och vektorer som är sätt att lagra värden på. Likaså ligger mycket fokus på kontrollstrukturer som används för att styra ett program utifrån olika förutsättningar och som ofta bestäms av användaren av programmet. Och du kommer lära dig att använda metoder som är en förutsättning för att skapa en kod av bra kvalitet. Koden du skriver mäts inte bara utifrån vad den åstadkommer utan hur bra du skrivit den. Att skriva en bra kod kräver träning och kunskap, och vi ägnar mycket utrymme åt att skriva om och resonera runt vad som kan sägas kännetecknа en bra kod.

Läroboken lutar sig mot principen objektorienterad programmering som ofta kallas för ett programmeringsparadigm. Framåt slutet av läroboken kommer dina kunskaper och förmåga att vara redo att ta dig an detta, och då öppnas den nya världen som programmering erbjuder upp på riktigt.

Boken är skriven utifrån gymnasieskolans läroplan och de riktlinjer som ligger till grund för kursen Programmering 1 (PRRPRR01).

ISBN 978-91-7531-165-4
9 789175 311654 >

DOCENDO