



MALMÖ HÖGSKOLA

Fakulteten för Teknik och samhälle

Programmering med C#, grundkurs

Ditt första C# program

[Farid Naisan](#)

Universitetsadjunkt
farid.naisan@mah.se

sep 2013

Innehållsförteckning

Kom igång – bli produktiv från dag 1!	3
1. Ett C# program.....	4
2. Skapa ett konsolprojekt.....	4
3. namespace	14
4. Klassdefinition och dess synlighet.....	15
5. Dokumentation med kommentarer	16
6. Instansvariabler eller fält, attribut eller medlemsvariabler	17
7. Metoder	20
8. Skapa en instans (ett objekt) av en klass.....	21
9. Kompilera och testköra	22
10. Sammanfattning	23

Kom igång – bli produktiv från dag 1!

Att utveckla ett C# program kräver egentligen inga avancerade verktyg och vill man ta den hårda vägen det räcker med Windows Anteckningar (Notepad) eller annan textbehandlare, tillsammans med .NET SDK. Men det är knappast lönt eller praktiskt att inte utnyttja de moderna verktyg som idag finns till vårt förfogande. Visual Studio.NET (VS) eller Visual C# Express är enastående exempel på sådana verktyg. Förutom en massa tjänster så tillhandahåller de en miljö där programkod skrivs, kompileras, länkas, felsöks och paketeras för installation. Programmens inbyggda textbehandlare är utomordentligt anpassad till programutveckling och programmerares behov. Ett känt förkortat namn för utvecklingsmiljöer av denna typ är **IDE** som står för "Integrated Development Environment".

VS och även i expressversionen är utrustad med Microsofts teknik som går under namnet Microsoft IntelliSense®-tekniken. Så fort du skriver något i IDE:ns kodvy visas förslag till autokomplettering med ett antal ord eller fraser som tekniken antar vara relevanta. Detta ökar vägledningen för programmeraren och minskar kodskrivningsarbetet samtidigt som det minimerar risken för felstavning. Det är många typer av applikationer som VS stödjer och det är relativt lätt att utveckla programmet oavsett applikationstyp. En applikation är en mjukvara, ett program som körs på en dator eller en annan enhet såsom mobiltelefon. Numera används ordet "app" eller "appar" väldigt mycket och trots att ofta är menas ett program som skall köras i mobiltelefonen, är en "app" en förkortning för en applikation.

I denna grundkurs jobbar vi med följande två typer av applikationer:

- Konsolapplikationer
- Windowsapplikationer

Andra typer av program såsom dynamiska klassbibliotek webb- och nätverksbaserade applikationer lämnas åt fortsättningskurserna. Konsolapplikationer saknar grafiska komponenter och jobbar endast med indata och utdata av typen text. Programmet körs från ett kommandofönster i en sekventiell ordning. Programmeraren har större möjlighet att styra körningssekvensen. Konsolapplikationer används

ofta för testning och för program som inte har mycket interaktion med användaren. Det är också lämpligt att börja med konsolapplikationer för att lära sig grunderna i programmering.

Windowsapplikationer innehåller grafiska komponenter av olika slag. Det är mycket vanligt att referera till ett grafiskt användargränssnitt som **GUI** (Graphical User Interface). I vilken ordning programmet körs bestäms för det mesta av användaren. Denna typ av program dominerar och idag innehåller även de enkla programmen grafiska användargränssnitt.

1. Ett C# program

Att utveckla en applikation i ett OO språk är att skriva ett antal klasser och låta dem använda varandras tjänster. Ett litet program kan bestå av en enda klass, fast det är mycket vanligare med program bestående av flera klasser. En större applikation kan innehålla uppåt flera tusen klasser. Även små program har ofta många klasser. För att skriva ett program, skriver man klasser. En klass utgörs av data som kallas för instansvariabler, fält, attribut, eller medlemsvariabler, samt operationer som kallas för metoder eller funktioner. Ett objekt är ett unikt exemplar av en klass, d.v.s. en instans av en klass.

I denna beskrivning skapar vi en konsolapplikation steg för steg. Vi använder VS IDE för att skapa projektet i, men förfarandet är mestadels detsamma även i expressversionen av Visual C#.

Som ett övningsexempel skriver vi ett program som hanterar en produkt, dess namn, pris och ett produktnummer.

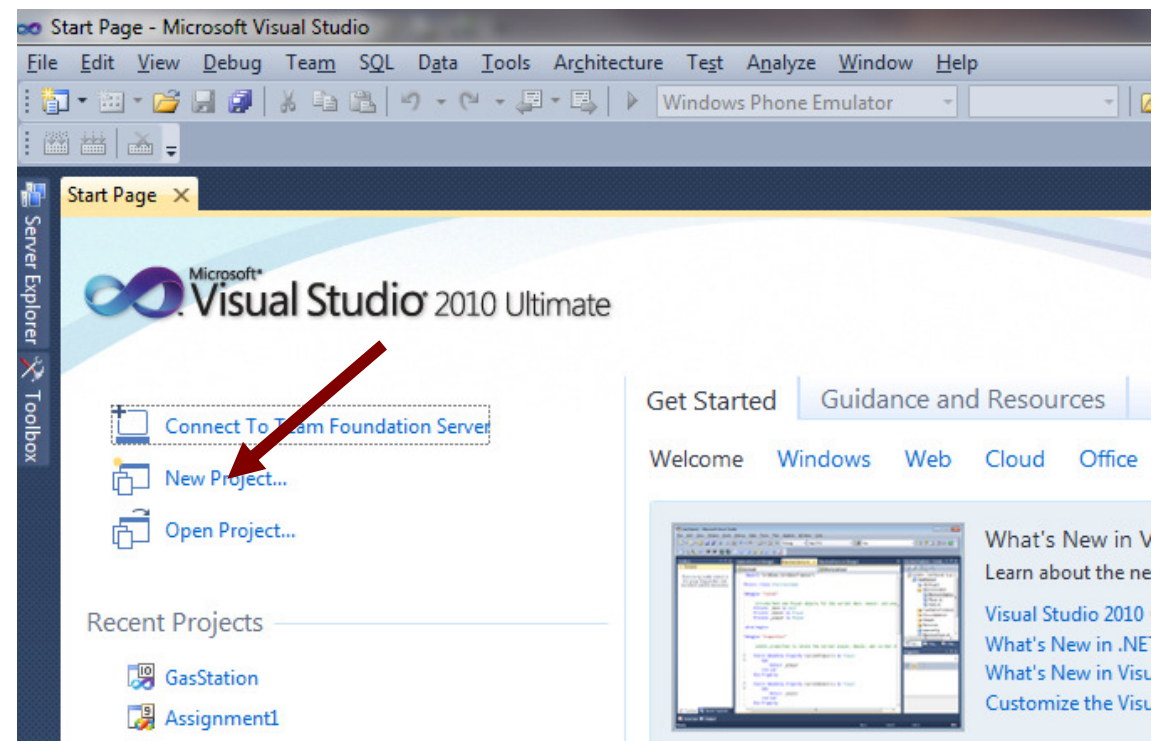
2. Skapa ett konsolprojekt

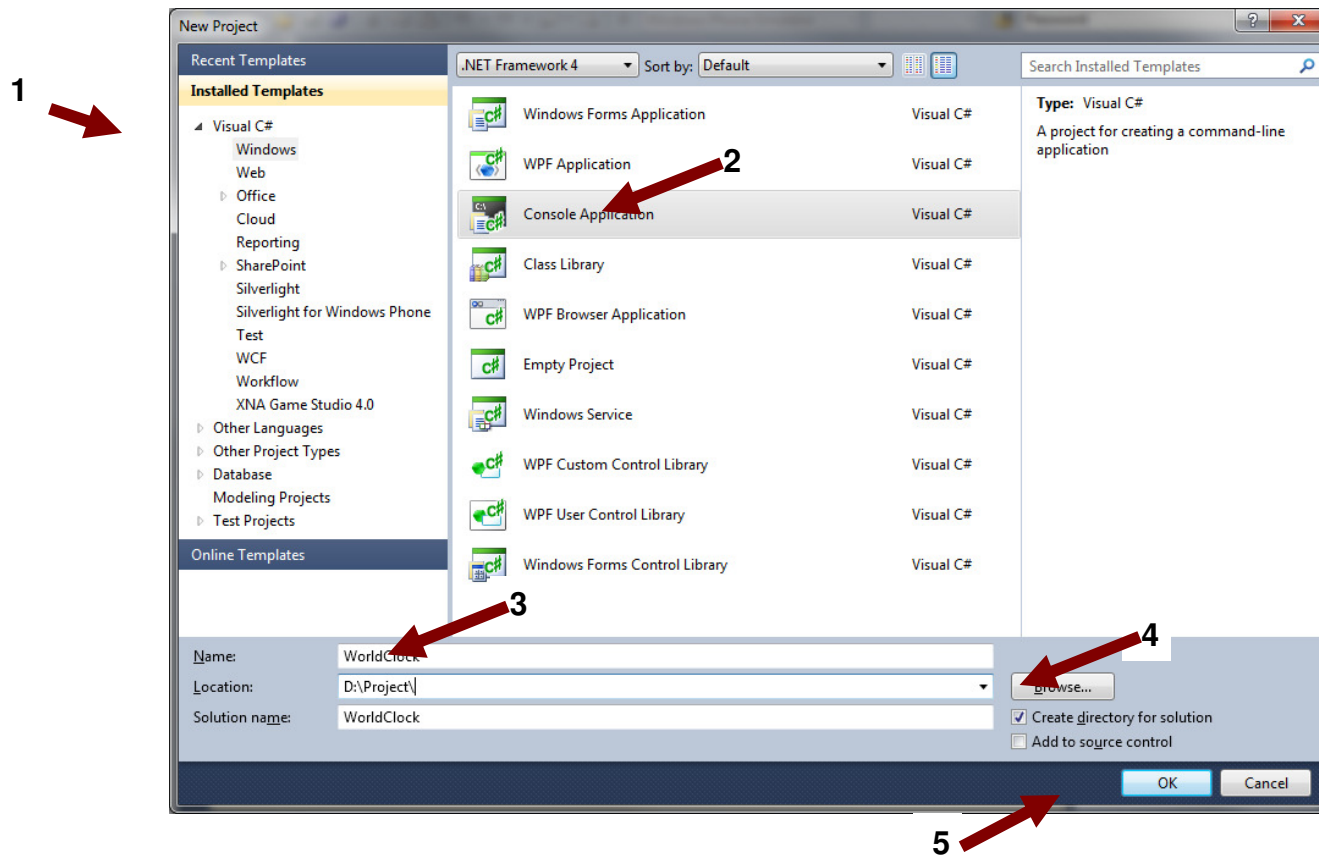
Demonstrationen nedan görs i Visual Studio 2010 men det fungerar ungefär på samma sätt även VS 2012. Du kommer att känna igen förfarandet även om du använder Visual C# 2010 Express eller en senare version av VS. Vi skall skapa en applikation som visar aktuellt datum och aktuell tid. I denna övning, nöjer vi oss att visa datorns system datum och tid.

Starta Visual Studio och välj New Project från startsidan för att skapa en ny applikation. Open Project används förstås på ett tidigare sparat projekt.

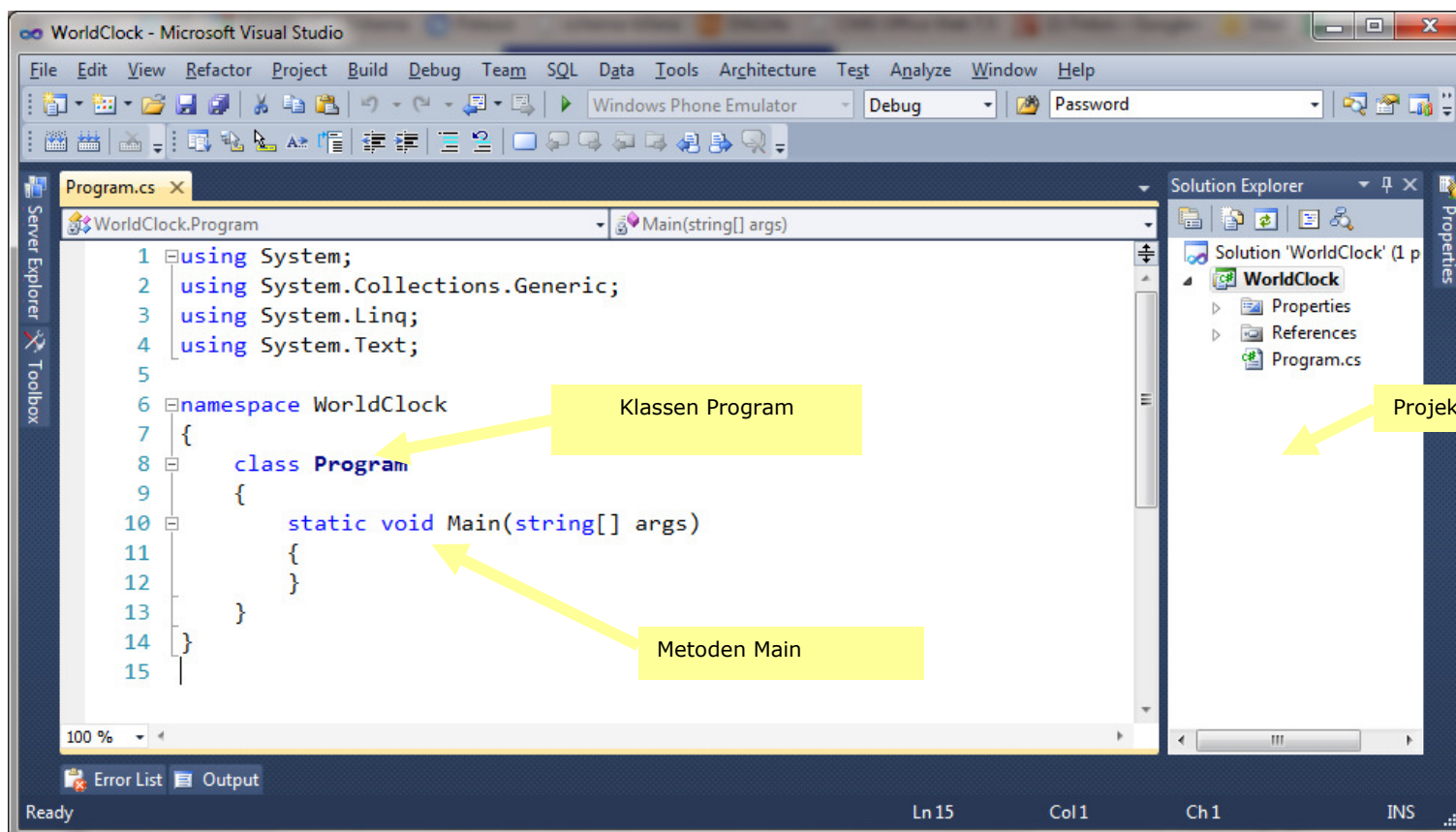
En dialogbox (nedan) kommer upp. Följ stegen nedan:

1. Välj språket Visual C#, och typen Windows som typ av projekt.
2. Välj typ av applikation, Console Application.
3. Välj ett namn för applikationen.
4. Bläddra till den enhet och mapp du vill skapa projektet.
5. Tryck OK.





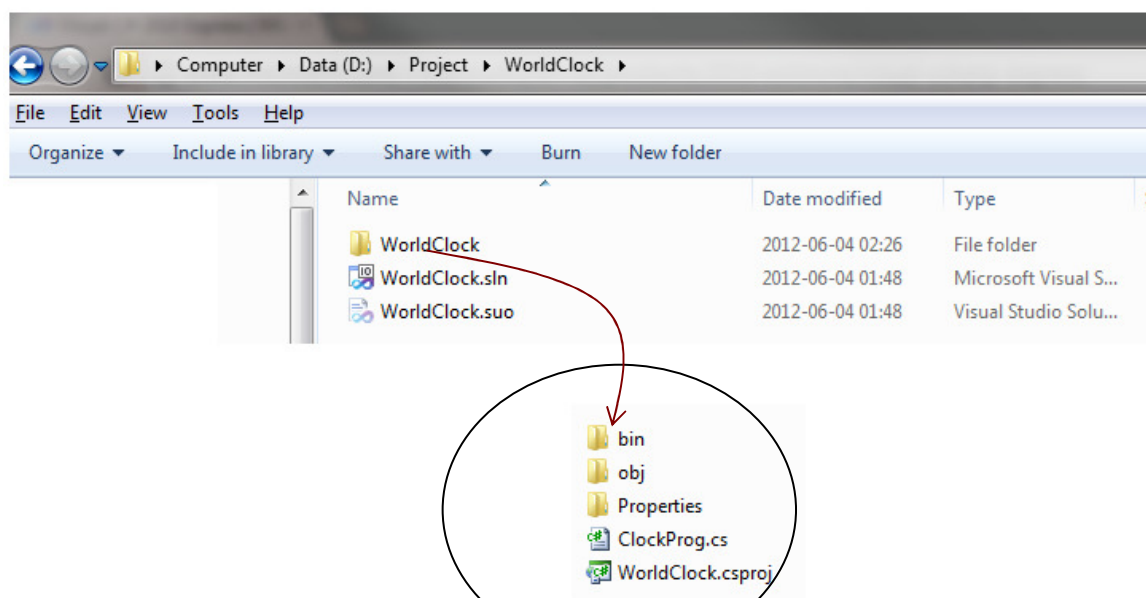
IDE:t skapar en s.k. .NET Solution som är i princip ett gruppnamn för de projekt som skall ingå i din lösning. Du kan alltså skapa flera projekt i samma solution. I vissa versioner av VS (t ex Visual Studio Ultimate), kan du behöva slå på detta val via menyn Tools, Options. I denna solution har vi än så länge bara ett projekt, nämligen **WorldClock**. Visual Studion skapar också en klass för dig och kallar den för Program.cs. Denna klass är förberedd som start klass.



I lösningsutforskarfönstret (Solution Explorer) längst upp till höger är alla ingående filer och komponenter listade. Mappen **Properties** innehåller filer som har att göra med assemblyn (själva applikationen). **References** är en mapp som lagrar referenser till alla de komponenter och assemblies som denna applikation använder sig av, t.ex. DLL:er från .NET Framework. **Program.cs** är en C# class som IDE:n skapat och förberett för oss. Denna klass har en metod med namnet Main som är deklarerade med nyckelorden **static void** och det blir just denna metod som kommer att automatiskt köras när programmet startas. Main-metoden är alltså entrén till programmet. Namnet

”Program” är rätt abstrakt och säger inte mycket om vad klassen representerar. Det är därför en bra idé att döpa om klassen till något vettigare och mer informativt. Vi väljer namnet **ClockProg**. För att genomföra namnändringen, högerklicka på **Program.cs** i solution explorer och välj i menyn Rename. Skriv in ClockProg.cs. Visual Studion kommer att fråga dig om du vill att VS skall göra ändringen också i koden på all de ställen namnet Program användes före ändringen. Du skall tacksamt ta emot erbjudandet och svara ”Yes”.

Nu, gå in på den mapp där du har skapat projektet och kolla filstrukturen. Du kommer att märka att VS har skapat samma mappstuktur som i projektutforskaren. Projektutforskaren visar det logiska filstrukturen medan i Windows utforskaren ser du den fysiska filstrukturen.



Mappen **bin** och **obj** är för kompilersändamål men de också innehåller slutprodukten, dvs. programmet, exe assemblyn efter kompilering. Mappen Properties är en viktig del av projektet. **Sln**-filen sparar solution data och **csproj** sparar allt om projektet. För att öppna projektet nästa gång, är det **sln** filen som man startar med en dubbelklickning. Om sln-filen av någon anledning inte fungerar, kan applikationen också startas med **cproj**-filen.

IDE:t har också skrivit skalet till metoden **Main** eftersom varje C# program måste ha en klass som innehåller metod Main. Denna metod talar för kompilatorn att här börjar programmet. Det går att ha flera klasser innehållande metoden Main (och detta görs t.ex. i testningssyfte) men då måste en klass med Main metod väljas som startobjekt via projektgenskaper (klicka på Properties, välj Open).

Vi har nu en startklass som startar programmet. Vi kan stoppa all kod här men vi vill från början göra saker på rätt sätt. Ett första läxa att lära sig här är: "Dela ansvar!" Varje klass skall göras ansvarig för ett visst område. Startklassen, **ClockProg**, låter vi vara en "startklass". Vi skapar en ny klass för att ansvara för datum och tidvisning. Obs. det är mycket som på engång är nytt för dig men du behöver inte lära dig och förstå allting nu. Vi återkommer till dessa och jobbar med dem igen och igen. Vi skall skapa en klass "**Clock**" för att hantera allt om datum och tid. **Clock**-klassen kommer att användas av **ClockProg**. Låt oss skapa klassen Clock nu. Högerklicka på projektnamnet **WorldClock**, välj **Add** och sen **Class** som i bilden. Skriv in namnet **Clock.cs** i längst ner i rutan i den dialogboxen som visas i VS.

Komplettera klassen som visas i skärmbilden nedan.

Vad behöver du veta vid detta stadium:

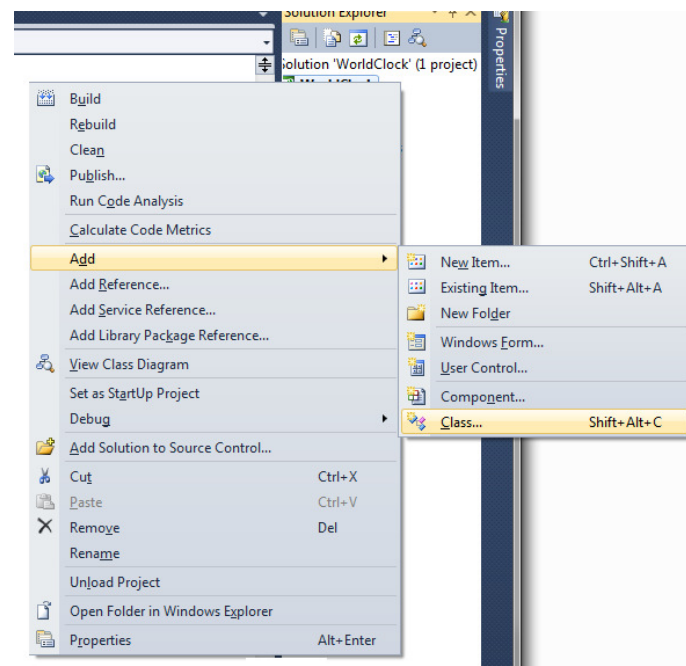
- **ClockProg** kan inte se och nå instansvariabler och metoder som är privata (deklarerade med ordet private).
- **ClockProgram** kan endast se och nå den metod som är publik.

Tumregel 1: Låt alla instansvariabler vara private-deklarerade.

Tumregel 2: Låt endast de metoder som behöver exponeras till andra klasser (för de erbjuder tjänster) vara public-deklarerade. Metoder som används internt skall vara private-deklarerade.

Bägge av dessa regler följs i klassen **Clock**.

Klassen **ClockProg** visas nedan kompletterad med några kommentarer.



```
1 //Code Example
2 //Clock.cs
3 //By: Your name and date
4
5 //System is a namespace in .NET Framework that hosts a
6 //great number of ready to use classes. This namespace
7 //includes classes that are needed for common tasks such
8 //as input/out actions.
9 //The keyword "using" says to the compiler that we will
10 //be using some classes from the namespace (here from
11 //the System).
12 using System;
13 using System.Collections.Generic;
14 using System.Linq;
15 using System.Text;
16
17 //VS has also assigned a namespace for our project, WorldClock
18 //too.
19 namespace WorldClock
20 {
21     //The class definition - syntax
22     public class Clock
23     {
24         //here we can define instance variables for the data and methods to do things (with or
25         //without the data saved in teh instance variables.
26
27         //Declare a variable to store a city name for which we will show the time. This info
28         //comes from the user (user = person that runs the program)
29         private string city;
30
31         //Now we do the following: 1. Ask the user for the city name
32         //                          2. Show the time for the user.
33
34         //All code we write, except instance variables and comments must be contained within a method!!!
35     }
```

```

36 //WhatIsTheTime is a method that I write to do the above.
37 public void WhatIsTheTime()
38 {
39     //Delegate the job to two other smaller methods
40     //Let the user to give a city name - call method one:
41     ReadAndSaveCityName();
42
43     //call method two:
44     ShowDateAndTime();
45 }
46
47 //Method to read user's input - the name of the city.
48 //Save the input in the instance variable "city" as defined (declared above).
49 private void ReadAndSaveCityName()
50 {
51     //Clear the screen
52     Console.Clear();
53     Console.WriteLine("Hi, what is the name of the city?");
54
55     //When the use has entered a text and pressed the Enter key,read the given text
56     //and save it in this object
57     city = Console.ReadLine();
58
59 }
60
61 //Now write the current date and time.
62 private void ShowDateAndTime()
63 {
64     Console.WriteLine("***** ABC - Apu's Big Clock *****");
65     Console.WriteLine("    The date in " + city + ": " + DateTime.Now.ToLongDateString());
66     Console.WriteLine("    The time in " + city + ": " + DateTime.Now.ToLongTimeString());
67     Console.WriteLine("*****");
68 }
69
70 }
71 }

```

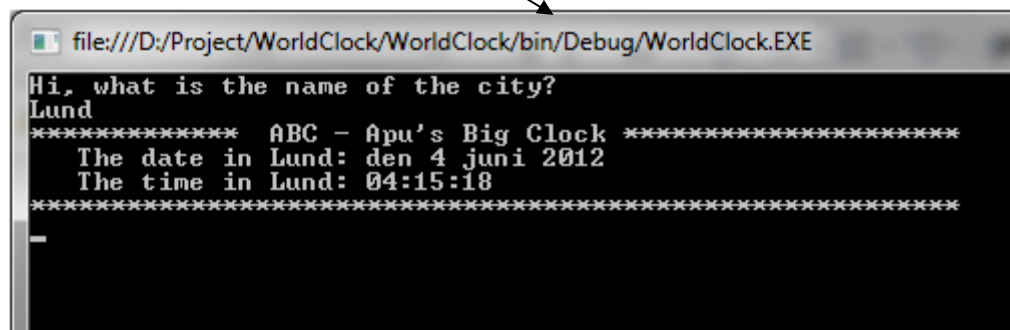
```
1 //Code Example
2 //ClockProg.cs
3 //By: Your name and date
4
5 using System;
6 using System.Collections.Generic;
7 using System.Linq;
8 using System.Text;
9
10 //This class is the start class and its method Main will be the first code
11 //that will be executed at run time.
12
13 //The namespace of our project. All other classes hat are defined in othe files but
14 //with this namespace declaration will find and use each other. Spelling is important!
15 namespace WorldClock
16 {
17     //Class name declaration
18     class ClockProg
19     {
20         //The Main - the first method that gets executed at run time
21         static void Main(string[] args)
22         {
23             //To do:
24             //1. Declare and create en object of the class Clock
25             Clock clockObj = new Clock();
26
27             //2. Call the clockObj's public method to show date and time
28             clockObj.WhatIsTheTime();
29
30             //The Prompt window will disappear so fast you will not notice it.
31             //We can keep the window by a small trix: put the console in the
32             //input mode so it waits for input
33             Console.ReadLine(); //to close the window press the Enter key.
34
35         }
36     }
37 }
38
```

Nu har vi skrivit färdig klassen `Clock` (sparad i filen `Clock.cs`), men klassen är bara en definition och den gör inget tills vi skapar ett objekt av den och anropar (använder) dess publika medlemmar. Läs kommentarerna i koden (bilden nedan):

Vad vi har gjort nu är att vi har kopplat **ClockProg** till klassen **Clock**. **ClockProg** använder ett objekt av **Clock** men objektet vet inget om **ClockProg** och helst så det skall fungera mellan två klasser: en klass skall agera som "Försäljare av tjänster" och den andra klassen skall agera som "Köpare av tjänster". Var och en av dessa kan ha motsats roll i förhållande till andra klasser.

Kompilera och kör (tryck knappen F5). Om du skrivit exakt som i koden ovan och inte stavat fel, bör du se ett resultat som i bilden.

Obs! sökvägen till exe-filen



```
file:///D:/Project/WorldClock/WorldClock/bin/Debug/WorldClock.EXE
Hi, what is the name of the city?
Lund
***** ABC - Apu's Big Clock *****
The date in Lund: den 4 juni 2012
The time in Lund: 04:15:18
*****
```

Nedan följer en mer detaljerad förklaring av ett antal begrepp som berörts i detta exempel.

3. namespace

Den allra första raden i klassfilen innehåller satsen `using System;`. I .Net grupperar man sina klasser i olika logiska grupper och väljer ett gruppnamn för dem för att undvika namnkonflikter. För att kunna använda en klass i en grupp måste man skriva gruppnamnet tillsammans med klassnamnet, t.ex.

```
System.Console.WriteLine ("Hej!");
```

För att slippa skriva namespace-namnet hela tiden, talar man en gång för hela filen att de ingående klasserna (i aktuell fil) använder klasser från en viss samling.

```
using System;
```

Man behöver inte längre skriva System när man jobbar med klasser i namespacet.

```
Console.WriteLine ("Hej!");
```

En namespace kan nästla flera andra namespace.

```
using System.Collections.Generic; //Har tre nivåer
```

System är ett gruppnamn i .NETs klassbibliotek som omfattar en massa färdiga klasser .NET gruppen på Microsoft skrivit för vår användning. En sådan klass är **Console** som har metoder som underlättar inmatning och utmatning av data från och till ett konsolfönster. Gruppen kallas tekniskt för **namespace**, och i .NET handlar det bara om en logisk gruppering och klasserna behöver inte alls ha samma filstruktur fysiskt i datorns hårddisk. För varje projekt som vi skapar skall vi välja och använda en **namespace**, alltså ett namn som vi själva kommer på. Visual Studio väljer dock projektnamnet som namespace namn men du kan ändra det genom att högerklicka på **Properties** i projektutforskarfönstret, välja **Open** och där hittar en ruta för det. Klasser som ingår i samma **namespace** har lättare att hitta varandra.

4. Klassdefinition och dess synlighet

En klass börjar med det reserverade ordet **class** följt av programmerarens namn på klassen.

```
class ClockProg
```

I vårt exempel ovan byggde två klasser, ClockProg och Clock. Beroendet mellan dessa två satte vi så att ClockProgram använde sig av Clock, eftersom Clock klassen definierar tjänster som ClockProg behöver använda.

För att skapa en ny klass börja med att skapa en tom textfil och spara den med cs ändelse. Sedan gäller det att definiera klassen och skriva kod. Det är dock lättare använda IDE:ns class-wizard. Högerklicka på projektnamnet i projektutforskaren, välj Add och klicka på klass längst ner i menyn. Ange ett filnamn och spara i samma mapp som de andra cs-filer.

I C# är klass en typ, d.v.s. en datastruktur. Mycket av det som programmeras skrivs som kodblock med en start- och en slutmarkering. En klass är ett sådant block, och ska enligt språkets regler (syntax) ha en start- och slutmarkering med hjälp av klammar parenteser '{' och '}'. Lägg märke till att även namespace deklARATIONER innehåller en startparentes (längst upp) och en slutparentes (längst ner).

För att en klass kunna användas av andra klasser behöver den vara synlig och åtkomlig. För synligheten har språket några reserverade ord som heter modifierare. För tillfället räcker det att bekanta sig med **public** och **private**. En **public** modifierare gör en deklaration (definition) synlig och åtkomlig för omvärlden så att andra klasser kommer åt dem utan någon begränsning, medan en **private** enbart kan nås av klassen själv men blir osynlig och oåtkomlig för andra klasser. Privata attribut och privata metoder nås av klassen själv men inte av andra. Privata attribut och publika metoder är grunden för inkapsling.

Klasser brukar behöva vara publika för annars är det meningslöst att ha en privat klass då det inte går att skapa ett objekt av det utifrån. Den enda gång man kan tänka på användning av en privatklass är då den är nästlad i en annan klass. Om du inte skriver private eller public framför klassdefinitionen, blir klassen automatiskt åtkomlig och synlig inom dess namespace. Modifieraren för den här typen av

åtkomlighet heter **internal** i C#. Det är också helt OK att deklarera klasser som **public** utan att bryter mot OOP lagar. Det som är viktigt att komma ihåg är att du antingen använder public eller ingen modifierare alls konsekvent för alla klasser i namespace.

```
namespace WorldClock
{
    public class Clock
    {
    }
}
```

5. Dokumentation med kommentarer

Det är viktigt att koden är så tydligt som möjligt. Detta kan delvis göras genom att använda vettiga namn, indentera koden, och skriva tydliga satser samt kommentera koden. Namn på variabler och metoder och även filer och projekt bör vara tillräckligt långa och ska antyda syftet med dem. Dessutom skall kod kommenteras genom med att skriva kommentarrader direkt ovanför eller framför en sats. Kommentarer är bara anteckningar för oss själva (och andra som granskar koden) och därför kompileras. Det finns minst tre sätt att kommentera i C#:

```
// Sätt 1: all text efter symbolen '//' blir kommentar.
/* Sätt 2: att kommentera mitt i en kodrad eller att kommentera flera rader
   på en gång, en helfil kanske.*/
/// Sätt 3: som sätt 1 men det är anpassat till XML. Kompilatorn kan då
/// tolka vissa taggar och göra en fil i XML-format som sen kan även omvandlas
/// till html. I Visual Studio görs även en automatisk koppling till
/// mellan XML kommentarer och Intellisense.
```

Kommentarer hoppas över av kompilatorn vid tolkning till maskinkod och därför ökar de inte assemblyns storlek. Syftet är att göra koden lättläst, lätt att förstå och därmed lättare att underhålla och minska fel och därmed spara tid och resurser.

6. Instansvariabler eller fält, attribut eller medlemsvariabler

En produkt-klass behöver spara data om ett visst produktobjekt för att kunna manipulera dem. Data kan vara input, mellanberäkningar eller output för redovisning. Vilka data som skall gälla beror på uppgiften och för vilket ändamål produkten används. Här bestämmer vi oss för produktnamn, produktnummer och inköpspris för en produkt.

Det reserveras ett minnesområde i primärminnet för varje program under körning, där programmets instruktioner och data lagras under körningstiden. När programmet avslutas befrias minnesutrymmet. Ett objekt av en C# klass utgör del av instruktionerna och det kan innehålla data. Objektet får ett minnesutrymme så fort det skapas med ordet new.

För att dator skall kunna allokera (reservera) rätt mängd primärminne till klassen och dess data måste det framgå i många fall redan vid kompileringen. För att spara t ex produktpriset, behöver datorn veta om priset är en 16, 32 eller 64 bitars heltal eller ett 32, 64 eller 128 bitars reellt tal för att allokera ett tillräckligt stort antal bitar för värdet. Var i minnet operativsystemet allokerar minne är mindre intressant för oss som programmerare, vad vi behöver veta är hur vi kan nå värdet. Detta görs med hjälp av en variabel som är ett namn som programmeraren väljer för att referera i koden till minnesutrymmet. En variabel är en behållare av data och representerar en plats i minnet där variabelns värde är sparat.

För de tre egenskaper som nyss nämndes, nämligen namn, nummer och pris så behöver vi ha tre variabler (tre attribut eller fält). Sättet att tala om detta för kompilatorn kallas för deklaration. Det finns två typer av variabler:

- Variabler av värdetyp,
- Variabler av referenstyp.

6.1. Värde-variabler är enkla primitiva typer av data som kan lätt grupperas i tal, text och logiska variabler.

Tal kan vara heltal som 5, -23456, 0 och eller reella tal som -0.5, 0.0, 12.55678E4. För tecken (bokstäver och symboler) följs Unicodes teckentabell, t.ex. 'A', '5', '+', 'ö' och 'Å'. Exempel på text är "Hallå, hur mår du?", "34.5", "5". Det finns en annan viktig typ av data som varken är tal eller text utan de har ett logiskt värde som sant eller falskt, ja eller nej, på eller av.

Programmeringsspråk brukar ha inbyggda typer som man kan skapa variabler av. I C# finns många inbyggda datatyper. För heltal finns det bland annat **int** och **long**, för reella tal finns **double**, **decimal** och för logiska värden heter typen **bool**. För tecken har C# typen **char** och för texter (som är en sekvens av tecken) används **string** som i själva verket är en referenstyp men hanteras i många fall som de andra primitiva datatyperna.

En variabel av typ **int** sparar heltalsvärden medan en variabel av typen **double** används för reella tal. När ett värde kan förekomma i både heltals- och bråktalsform, går det att omvandla bråket till ett närmast heltal eller använda en variabel av **double**. Ett tal som 2 sparas då som 2.00.

Ett produktnamn är ganska självklart en text och därför använder vi typen **string**. Vi bestämmer att produktnummer skall vara ett heltal och därför kan variabeln deklarerars som **int**. Inköpspriset kan ha decimaler så vi använder typen **double**.

Nästa viktiga steg är att bestämma om variablerna skall vara tillgängliga för andra klasser, d.v.s. vi får bestämma oss för **private** eller **public** modifierare. Om vi låter variablerna vara publika, kan vilken metod som helst i vilken klass som helst ändra värdet på variabeln utan kontroll. Det här är mot en av OOPs viktiga grunder, inkapsling och skall undvikas och får inte användas på kursen. Som tumregel så deklarerar alltid attributen privata om det inte finns någon särskild anledning för att de ska vara publika.

6.2. Referenstyper är variabler som har objekt som typ; de behåller inte sitt värde direkt men de vet var i minnet objektet finns. Mer om detta kommer senare i kursen.

Nu när vi har gått igenom begreppen, kan vi deklarera klassen **Product** och dess fält.

```
using System;

namespace Products
{
    public class Product
    {
```

```
//attribut (fält, medlemsvariabler)
private string namn;    //deklaration av variabeln name för att spara produktnamn
private int nummer;     //id nummer för produkten
private double pris;    //Inköpspris

//metoder

} //class

} //namespace
```

Observera syntaxen och lägg märke till att deklarationen börjar med synlighetsmodifieraren (**public**, **private**, etc) följt av datatypen (**string**, **int** eller **double**), ett eget namn och slutligen ett semikolon tecken som aldrig skall glömmas. Variabeln **nummer** till exempel får en plats i primärminnet som är 32-bitars stor där det går att spara väldigt stora heltal. Varje objekt som skapas av denna klass kan ha egen uppsättning av värden på namn, nummer och pris för det aktuella produktobjektet. Det finns förstås ett antal regler på namngivningen som språket sätter. Namn

- kan innehålla siffror men kan inte börja med en siffra,
- kan inte vara identiskt med reserverade ord,
- kan inte innehålla mellanslag eller symboler som +, -, punkt, kommatecken, etc.,
- kan vara rätt långa.

När ett objekt skapas av klassen (instansieras) nollställer kompilatorn värdet av instansvariablerna. Variablerna i klassen **Product** är deklarerade privata vilket innebär att andra klasser inte kan nå dem (tack o lov). Men hur kan andra objekt ändra värdet av variablerna (ge input) eller hämta värden för sina andra operationer? Jo, det finns en special metod som heter Property som används i sammanhanget. Detta behandlas i en senare modul.

7. Metoder

Klassen vi har skrivit har definierat datavariablerna men det är inte mycket som kan göras med dem. För att klassen skall kunna manipulera sina data och eller erbjuda tjänster till andra klasser behöver den ha metoder. Vi skriver en metod som skriver ut de aktuella värden som är sparade i variablerna.

En metod börjar också med en synlighetsmodifierare, **public** eller, **private** (det finns andra modifierare också). Eftersom metoder utför operationer som andra klasser kan beställa, brukar de ofta vara synliga, d.v.s. publika. Det reserverade ordet **void** talar om för kompilatorn att metoden utför operationer utan att returnera något värde. **WhatIsTheTime** i vår **WorldClock** exempel är vårt namn på metoden. Metoddefinitionen avslutar med ett tomt parantespar som betyder att inga in-variabler kan eller ska skickas in i just denna metod (metoden tar inga in-argument). Som allt annat är även en metod ett block av kod vars start och slut markeras med ett par matchande klammarparenteser ('{' och '}'). Det är vanligt att starta metodnamnen med stor bokstav i .Net världen.

En metod kan vara antingen **void** eller en metod med **returvärde**.

```
//Metod som räknar och returnerar momsen
public double CalcMomsen()
{
    return 0.25 * pris;
}
```

Denna metod skiljer sig från de **void** metoder vi skrivit tidigare nämligen att den returnerar ett **double** tal. I övrigt fungerar den på liknande sätt, d.v.s. den utför ett antal operationer. Returvärdet i detta fall är en **double** men hade kunnat vara av en annan typ så som **typ, int, long** eller av en objekttyp om vi hade velat det istället. Metoden kommer att returnera värdet efter ordet **return** vilket här blir resultatet av beräkningen 0,25 multiplicerat med värdet i variabeln pris.

Klasser kommunicerar med hjälp av metoder. De anropar metoder av de objekt av klasser som de använder sig av. Metoder på objektet som är deklarerade (definierade) med **void** i sin klass anropas (aktiveras) som nedan:

```
prodObj.SkrivutVärdena();
```

När en metod anropas hoppar exekvering direkt till metoden och när metoden har körts färdigt återkommer exekvering tillbaka till samma rad och fortsätter med nästa sats. En metod som inte är **void** har ett returvärde som bör användas. Returvärdet kan tas emot i en variabel av samma typ som returvärdet, som i koden ovan, eller användas direkt i andra satser.

```
double moms = product.CalcMomsen();
```

När metoden **CalcMomsen** på objektet av typen **Product** körs, kommer värdet av det som returneras i metoden att hamna i variabeln **moms** i koden ovan.

8. Skapa en instans (ett objekt) av en klass

I vårt första exempel skapade vi en konsolapplikation med två klasser. Den första klassen **ClockProg** innehåller metoden **Main**. Den andra klassen heter **Clock** och en enda instansvariabel **city** och några metoder. Klasserna är sparade i var sin **cs**-fil. Att komma ihåg är att klass bara är en definition för objekt och för att kunna utnyttja en klass i andra objekt så måste vi **instansiera** ett objekt från klassen. Att instansiering betyder att skapa ett visst objekt av en klass. Ett objekt (en instans) skapas med det med instruktionen **new**. När vi vill vi skapa ett Clock objekt, det sker i två steg:

1. Deklarera en variabel för objektet vi vill skapa. Variabeln deklarerar på samma sätt som för variabler av värdetyp fast nu är typnamnet för objektet klassnamnet av den klass som vi vill skapa ett objekt ifrån.
2. Skapa objektet med **new** och bind objektet till variabeln (referensen till instansen lagras i variabeln)

```
Clock clockObj; //1.deklaration av referensvariabeln, objektet är ej skapat (clockObj pekar inte på något objekt)
clockObj = new Clock(); //objekt skapat, nu har man tillgång till objektets publika medlemmar
```

Stegen ovan kan göras på en och samma rad:

```
Clock clockObj = new Clock();
```

Nu kan vi använda objektet och köra dess metoder genom att använda objektreferensen: **clockObj**. En klockfabrik har en form för att tillverka flera av samma typ av klockor. Du köper ett exemplar som har en viss färg, ett visst pris och en viss storlek. Detta motsvarar **clockObj** medan formen motsvarar klassen **Clock**.

Inne i en metod är en deklarerad variabel alltid privat och kompilatorn tillåter inte någon synlighetsmodifierare alls. Variabeln kallas då för en lokal variabel. Detta gäller både värde- och referenstyper. I vårt exempel deklarerar vi en lokal variabel av klassen **Clock** inne i Main metoden i klassen **ClockProg**. Tills dess att ett objekt av en klass skapas med new så håller inte variabeln adressen av något objekt. I sådana fall har referensvariabeln ett värde som kallas null och som säger att objektet inte har tilldelats något objekt att referera till. I deklarationen:

```
Clock clockObj;
```

har clockObj ett värde null. När man vill att ett objekt befrias från minne så kan man sätta värdet till null.

```
clockobj = null;
```

9. Kompilera och testköra

I IDE:t finns meny **Build** för kompilering och länkning och **Debug** för körning (och felsökning). Knappen F5 gör allt på en gång. Vid eventuella kompileringsfel fås ett felmeddelande. Debug är en riktigt användbar funktion i VS som gör en programmerarens liv lättare. När programmet inte fungerar som det skal så är Debug ofta det enda sättet att söka efter var felet inträffar.

10. Sammanfattning

Ett program i C# består av ett antal klasser. Den klass som innehåller metoden Main fungerar som entré till programmet och metoden körs automatiskt av operativsystemet när programmet laddas.

En klass består av fält och metoder. Variabler måste deklarerars innan de används. Instansvariabler skall helst vara privata, d.v.s. deklarerars med ordet **private**. All kodning sker inne i metoder. Metoder utför ett antal operationer. En metod har en returtyp eller är **void**. Metoder som behöver exponeras till andra klasser skall vara publika d.v.s. deklarerars med ordet **public**. Koden skall innehålla dokumentation i form av kommentarer. Varje sats i C# avslutas med ett semikolon. Alla start- och slutparenteser måste matcha.

Det som omfattats av detta kapitel är en kort genomgång av de grundläggande begreppen för att snabbare komma igång. Det mesta kommer att behandlas djupare och mer detaljerad i senare delar av kursen.

Kom ihåg:

Det enda sättet att lyckas med programmering är att öva, öva och igen öva. Innan du bestämmer dig om att programmering är för svår så fråga dig hur många rader av kod har du skrivit. Ha tålamod. Det finns alltid en tröskel att gå över och det tar olika tid för olika individer. Försök att inte fastna på saker mer än en kort tid. Fråga andra och använd deras erfarenheter.

Ingen fråga är dum. Det är dumt att inte fråga!

Lycka till!



Farid Naisan

Kursansvarig och lärare