MALMÖ HÖGSKOLA
Fakulteten för Teknik och samhälle

Programmering med C#, grundkurs

Datatyper, variabler och deklaration

Del 1

Farid Naisan
Universitetsadjunkt
farid.naisan@mah.se

sep 2013

Innehållsförteckning

Datatyper, variabler och deklaration

1. Representation av data

Ett datorprogram jobbar med data som behöver sparas och uppdateras vid olika intervaller för beräkningar, bearbetning och manipulering eller redovisning. Data har olika former. Det finns tre grundtyper

1. numeriska,
2. logiska och
3. text.

Många andra typer kan härleddas med hjälp av dessa. För mer komplexa data räcker dock inte dessa typer och flera attribut behöver sättas tillsammans för att beskriva dem. Attributen kan vara av grundtyper eller andra komplexa typer. Klasser är ett bra exempel i detta sammanhang. Klassen **Date** (alternativt namn för **DateTime**) från C# är en typ som definierar datum av olika slag. Den består av medlemsvariabler som är av de tidigare nämnda typerna.

Datavärden sparas i datorns arbetsminne och eftersom datorn endast jobbar med binära tal så sparas ett värde som en serie av 1:or och 0:or. En bit kan endast vara 1 eller 0. En Byte består av 8 bitar och därmed kan rymma maximalt $\mathbf{2^8 = 256}$ stycken värden. Om alla bitar i en Byte är en etta $(11111111)_2$, blir det talet bitarna representerar ett maximalt värde på 256 i vårt decimala system (bas 10) och om alla bitar är nollor $(00000000)_2$, blir värdet 0. Det går alltså att spara 0 till 255, eller 1 till 256 alternativt -128 till +127 i en Byte; det gäller att bestämma sig när man deklarerar variabeln. En två-bitars långt minne rymmer 2^{16} = 65536 som kan spara tal mellan 0 65535 eller – 32768 till +32767, osv.

2. Datatyper

För att rätt mängd minne reserveras behöver datorn i förväg veta typ av data och storleken av det minne det kräver. Programmeringsspråk brukar definiera några inbyggda datatyper som tar olika storlek i minnet och kan spara olika typer av data och värden inom olika gränser. Definitionerna görs med hjälp av några nyckelord, till exempel **int** som tar 32 bitar (4 byte) i anspråk och kan spara både negativa och positiva tal inom gränsen -2,147,483,648 och 2,147,483,647. Beroende på ändamålet väljer man den typ som passar bäst för den typ av data man jobbar med. Vad som är viktigt att komma ihåg är att en programmerare måste tala om för datorn i förväg vilken typ av data man vill jobba med och värdet måste vara inom de gränser datatypen definierar.

```
short tal1;
int tal2;
tal1 = 22650; //Inget problem
tal1 = 33650;  //Mycket problem då den övre gränsen överskrids
tal2 = 33650; //Inget problem
```

I koden ovan tal1 som en short överskrider short:s övre gräns som är 32767. Datatypen int som är 32 bitar lång och kan rymma tal som är betydligt större (se Tabell 1) skulle lösa problemet.

I programmeringssammanhang behöver man vara noga med typ av värde som koden hanterar. Ett värde kan skrivas i former 2, 2.0 eller ’2’ men vart och ett tillhör olika typer. Värdet 2 tolkas som en 32-bitars heltal, 2.0 som en 64-bitars flyttal (tal med decimalpunkt) och ’2’ representerar tecknet ’2’ som upptar 16 bitar i minnet.

3. Heltal

Numeriska typer är antingen av heltalstyp eller av flyttalstyp. Ett heltal ett positivt eller negativt värde som inte har någon decimaldel. C# erbjuder flera datatyper med olika lagringskapacitet och minnesstorlek gjorda för olika ändamål. Tabell 1 sammanställer alla heltalstyper som finns i C#. I tabellen finns också typen char och bool som beskrivs senare. De typer som börjar med tecknet **’s’** (som står för ”signed” är sådana som kan innehålla både positiva och negativa värden i motsats till de som börjar med tecknet **’u’** (som står för ”unsigned”) som endast kan ta positiva tal samt noll.

Tabell 1: Heltalstyper samt char och bool

Typ	Räckvidd	Lagringsstorlek	Namn i .NET Framework
sbyte	-128 till 127	8 bitar	System.SByte
byte	0 till 255	8 bitar	System.Byte
char	U+0000 till U+ffff	Unicode 16-bitar	System.Char
short	-32,768 till 32,767	16-bit	System.Short

ushort	0 till 65,535	16-bit	System.UShort
int	-2,147,483,648 till 2,147,483,647	32-bit	System.Int32
uint	0 till 4,294,967,295	32-bit	System.UInt32
long	-9,223,372,036,854,775,808 till	64-bit	System.Int64
ulong	0 till 18,446,744,073,709,551,615	64-bit	System.UInt64
bool	true och false	32-bit	System.Boolean

C# tolkar automatiskt ett värde utan decimaltecken som t ex -5 som en int typ. Detta är en implicit typtilldelning som kompilatorn gör utan din inblandning. Vill du som programmerare att talet skal tillhöra en annan typ, behöver du gör en explicit konvertering. Detta kan göras på olika sätt. Ett enkelt sätt är att markera talet med en bokstav, som F, D, L, etc. Tal som skall betraktas som long behöver ha ett suffix L.

Exempel på giltiga heltal är:

```
-5          12456       1077            -324567         +77L
```

Exempel på ogiltiga heltal är:

```
-5.         +55.0       1245,8          SEK1500.
```

4. Flyttal

Ett flyttal, som brukar kallas reella tal i matematiken är ett tal som har en decimalpunkt. Tal av denna typ kan vara float, double eller decimal. Tabell 2 visar detaljerna. Exponentiella tal går också under denna datatyp.

Exempel på giltiga flyttal:

```
-5.         3F          77.35 1.0e-3        21.0E4          2500M
```

Talet -5. tolkas automatiskt som ett double (64 bitar) av kompilatorn, medan 3F är en float (32 bitar) och tecknet 'F' berättar om det för kompilatorn. Talet 2500M representerar datatypen decimal (128 bitar) p.g.a. tecknet M (som står för "money"), något som alltid förväntas av

kompilatorn när det gäller tal av denna typ. Exponentiella tal skrivs med tecknet 'e' eller 'E', t.ex. 21.0E4 som representerar $21*10^4$. Notera också att C# använder endast tecknet '**.**' som skiljetecken och inte komma. Tabell 2 sammanfattar flyttalstyper i C#.

Tabell 2: Värdetyper - flyttal				
Typ	**Ungefärlig räckvidd**	**Antalet signifikanta siffror (precisionen)**	**Lagrings-storlek**	**Namn i .NET Framework**
float	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$	7	32 bits	**System.Single**
double	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$	15-16	64 bits	**System.Double**
decimal	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$	28-29	128 bits	**System.Decimal**

5. Tecken

Denna typ representerar ett tecken i taget. Datorn sparar tecken internt med dess numeriska representation enligt standard tabeller. C# använder standarden Unicode där varje tecken sparas i två Byte i minnet. Ett tecken omslutas av två apostrofer i C# som i exemplen nedan.

```
'5'        '+'        'd'                '\n'
```

I C# finns vissa tecken som aldrig skrivs i klartext, t.ex. radslut, radbrytning, tabulatorsteg m.fl., men de ingår ändå i teckentabellen. I C# finns det ett antal så kallade "Escapesekvenser" som kan användas för dessa. Escapesekvenser är en kombination av ett bakåt snedstreckstecken och en bokstav. Här är ett par exempel:

```
'\n'       ny rad
'\t'       ny tabulator
```

Både apostrofer och snedstreck ingår som del av syntax i C# kod. Frågan är hur man skall göra när dessa behöver visas som ett tecken och inte tolkas som ett Escapesekvens tecken?. Så här gör man då:

```
char c1 = '\''        //c1 innehåller nu tecknet '
char c2 = '\"'        //c2 innehåller nu tecknet "
char c3 = '\\'        //c3 innehåller nu tecknet \
```

Det är många flera tecken som ingår i serien, men man klarar sig utan dessa och använda de färdiga objekt från .NET Framework som definierar alla tecken på tangentbordet. **Keys** är ett sådant objekt som definierar tangenter och **Environment** har vissa användbara egenskaper som är ett bättre val då man jobbar med objekt. För att åstadkomma en radbrytning finns det ett alternativt sätt i .NET, nämligen **Environment.NewLine**, Å andra sidan fungerar Escapesekvenser mindre bra med grafiska Windows Forms kontroller.

6. Logiska värden

Ofta behöver man spara data som är av typ ja och nej, på eller av och många gånger behöver man också veta om ett villkor är sant eller falskt. Man sätter villkor vid programmering för att kunna utföra vissa operationer. Man behöver t.ex. veta om ett värde sparat i en variabel som hamnar i en nämnare är ett tal större än noll för att undvika division med noll. I alla dessa fall sparas resultatet som true eller false. Datatypen heter bool i C# och kan endast lagra ett värde true eller false , vilka är reserverade ord i .NET.

```
if (tal1 > 0)
{
    //kod exekveras endast om resultatet av vilkort = true
}
```

Ett booleskt värde kan inte omvandlas till andra typer och andra typer kan inte göras om till ett bool. Man kan alltså inte tilldela en boolesk variabel ett numeriskt värde.

7. Text

En text är en sekvens av tecken. För att lagra text har .NET en objekt datatyp som heter string. En string kan vara tom (d.v.s. innehåller inget tecken) eller ha ett eller flera tecken.

```
string namn = "Nisse Hult";
```

Objektet namn får automatiskt många metoder från sin klass. Om du skriver namn och sedan en punkt i en kodfil i Visual Studio kommer du se de metoder och egenskaper som är anslutna till objektet namn. Med en egenskap menas en special metod som heter ”Property” i C# och det jobbar vi senare i kursen. Det

finns också ett färdigt objekt med samma namn, d.v.s. string eller String som har ett antal mycket användbara metoder och egenskaper med vars hjälp kan textsträngar manipuleras på olika sätt. Typen kommer att beskrivs i ett annat kapitel mer detaljerat Det ska noteras att text omslutas av ett start- och slut citationstecken. Ett ”h” är en string medan ett ’h’ är ett tecken. Exempel på textsträngar:

```
”5”        ”Hej hopp!”
```

Om ett citationstecken ingår i texten behöver man använda sig av ett extra snedstreckstecken före tecknet som i kodraden nedan:

```
string meddelande = "Hon sa: \"hej\" till mig";
```

Variabeln meddelande innehåller texten: Hon sa: ”hej” till mig. Om tecknet ’\’ självt ingår i texten så gör man så här:

```
string filNamn = "D:\\Document\\CSharp\\text.txt";
```

Strängen resulterar i sökvägen D:\Document\CSharp\text.txt efter kompilering. Det finns ett annat sätt för samma ändamål:

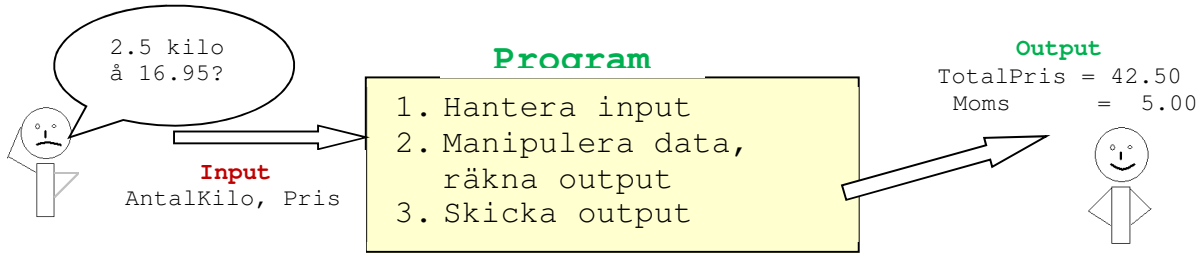
```
string filNamn = @"D:\Document\CShar\text.txt";
```

Tecknet ’@’ berättar för kompilatorn att alla tecken skall ingå i texten; annars skulle kompilatorn klaga över att t ex ’\C’ är ett escapesekvenstecken som den inte känner igen.

8. Vad är en variabel och vad menas med deklaration?

En **variabel** är ett namn på ett utrymme i minne som används för att behålla ett datavärde. En **deklaration** instruerar kompilatorn att reservera en del av minnet som är tillräckligt stor för att lagra en viss typ av värde och ger ett namn för att hänvisa till utrymmet.

Mycket av de data som hanteras i ett program sparas i variabler som behåller ett värde till dess värdet överskrivs. Tänk om du skulle skriva ett program som räknar belopp att betala för 2.5 kilo bananer som kostar 16.95 per kilo. Det är alldeles självklart att det inte är värt besväret. Det är helt enkelt inte någon mening att skriva program för ett visst smalt ändamål. Däremot vore det mer praktiskt att skriva ett program som räknar beloppet för en valfri produkt med ett valfritt pris per enhet och ett valfritt antal av produkten. Det är då det behövs variabler som skall behålla olika värden (av sin typ) vid olika tillfällen.



En del data behöver komma som indata från användare (den som kör programmet) till programmet. För dessa behöver programmet ha behållare, d.v.s. variabler som håller i värdena för att använda och processa dem senare. Programmet behöver också spara mellanvärden vid beräkningar och slutligen värden som skall lagras för redovisning som utdata från programmet till användaren. Data sparas då temporärt i minnet under olika tider dock längst tills programmet stängs. För att spara data permanent skall data sparas på permanent lagringsenheter såsom hårdisken, CD och DVD.

Kom ihåg att med ”värde” menas inte enbart tal utan även textsträngar och logiska värden. Du kommer väldigt ofta att jobba med data av typen string, d.v.s. texter. Lyckligvist, erbjuder C# många objekt som underlätter bearbetning av strängdata.

En variabel representerar ett visst utrymme på datorns minne. Den kan tänkas som en typ behållare för ett lagra värden. Behållaren kan spara en viss typ av data (heltal, flyttal, text, objekt eller ett logiskt värde som true eller false) och har en minimal och maximal gräns. Man kan dra liknelsen med t.ex. en mugg, en tekanna, en kastrull, en tallrik eller en sil. Samtliga har sina egenskaper och en volym. Man kan lagra te i en mugg men inte i en tallrik eller i en sil. Sedan kan man inte hälla en hel tekanna i en mindre mugg (medan på andra hållet går det bra). Försöker man göra det ändå så rinner det över (överflöde eller overflow).

Samma regler gäller för variabler som behållare av data. Ett ”överflöde” händer om försök görs att spara ett stort tal i en ”liten” variabel; frasen ”**floating-point overflow**” har väl många upplevt vid körning av program gjorda av slarviga programmerare. Att försöka spara en text i en variabel av numerisk typ blir inte heller en lyckad operation. C# kompilatorn är intelligent och kan upptäcka en hel del brister och fel redan under kompilering men vissa operationer är inte möjliga att upptäcka förrän programmet körs.

För att undvika alla sådana problem så spara rätt sak i rätt behållare och välja behållare av rätt storlek, därför kräver C# språket att man talar om det innan man använder en variabel. Man måste ange typen av variabeln (minnesstorlek och typen av data) samt ett namn för att referera till variabeln. Nedan är några exempel:

```
public class Product
{
    //Deklaration av variabler

    //input variabler (för indata från användaren)
    private const double moms = 12;  //procent
    private double aPris;          //a pris i en viss valuta
    private double antKilo;  //antalet kilo, kan vara flyttal

    //output variabler (för utdata till användaren)
    private double totalMoms;
    private double totalPris;

    //OPERATIONER = metoder

    //Steg 1: Läs indata från användaren och spara i variablerna ovan (input)
    //Steg 2: Utför beräkningen
    //Steg 3: Redovisa resultat (output)
  }
```

Nyckelordet private är en modifierare som sätter synligheten och åtkomligheten som ”endast i denna klass”. Ordet double är datatypen enligt tidigare beskrivning. Ordet **const** är ett nyckelord för att variabler som kan få ett värde endast vid initiering (moms = 12). Alla variabeldeklarationer måste avslutas med ett semikolon tecken.

9. Var och hur deklareras variabler?

Det är alltid inne i en klass där deklarationer av variabler blir aktuella och det är **fyra** lägen där deklarationer förekommer:

1. **Deklaration av fältvariabler (instansvariabler)** som görs inne i klassen i samma nivå som klassens andra medlemmar för lagring av klassens attribut. De kan vara typiska egenskaper som beskriver ett objekt av klassen. Attributen kan vara sådana värden som också utgör indata, utdata och mellandata. De värden som kan räknas fram av metoder behöver inte sparas i variabler. Metoderna kan då göra en beräkning varje gång metoden anropas (ShowBookInfo t.ex.).

```
class Book
{
    //Deklaration av fälten
    private int pages;      //värdetyp
    private string title;   //referenstyp (objekt)
    //. . .

    public void ShowBookInfo()
    {
        string textOut = String.Empty;  //lokal variabel med initiering

        textOut = String.Format("Bokens titel: {0}, Sidor: {1}", title, pages);
    }
}
```

Instansvariablerna som pages ovan är synbara och åtkomliga för alla metoder i klassen. De skapas vid skapande av ett objekt av klassen Book och lever så länge objektet lever. Ett objekt av klassen skapas med nyckelordet **new** som i kodexemplet nedan:

Book är en datatyp som vi har definierat själva (en klass är en typ).

```
class BookTest
{
    private Book nyBok = new Book(); //Deklaration och skapande av en variabel av typen Book

    // . . .

}
```

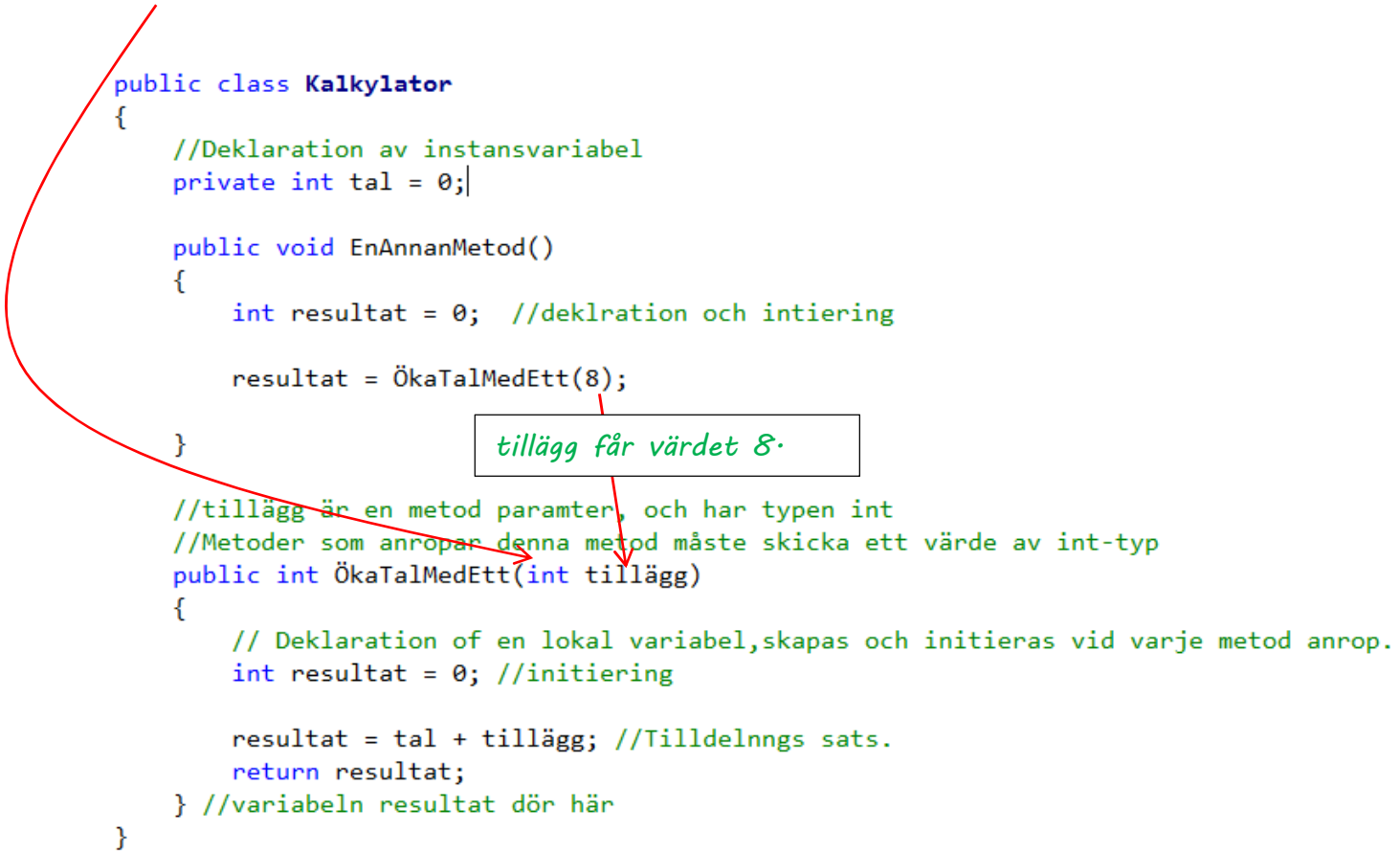
2. **Deklaration av lokala variabler** som görs inne i klassens **metoder** för att spara mellanvärden (temporära värden). Lokala variabler kan inte nås av andra metoder, varken i klassen eller utanför klassen. De kan inte deklareras med accessmodifierare som private och public. Lokala variabler lever endast under den tid metoden är under exekvering, d.v.s. tills körningen kommer till metodens slutklammarparentes. I klassen Book och metoden ShowBookInfo är textOut en lokal variabel som deklareras i metoden. C# kompilatorn kräver initiering av lokala variabler. Med initiering menas att en variabel skall tilldelas ett värde vid deklarationen. Variabeln textOut initieras till en tom sträng i ShowBookInfo-metoden. Här är ett exempel till där en lokal variabel (resultat) deklareras och initieras.

```
//tillägg är en metod paramter, och har typen int
//Metoder som anropar denna metod måste skicka ett värde av int-typ
public int ÖkaTalMedEtt(int tillägg)
{
    // Deklaration of en lokal variabel,skapas och initieras vid varje metod anrop.
    int resultat = 0; //initiering

    resultat = tal + tillägg; //Tilldelnngs sats.
    return resultat;
} //variabeln resultat dör här
```

3. **Deklaration av metodernas parametrar.**

   En metod kan ha ingen, en parameter eller flera parametrar. Metoden **ÖkaTalMedEtt**, i exemplet nedan, förväntar sig ett indatavärde från den anropande metoden **EnAnnanMetod** genom sin parameter **tillägg**. Typen av denna parameter måste deklareras i metodnamnet (int här). Den metod som anropar **ÖkaTalMedEtt** måste skicka ett **int**-värde, inte en **double, string, bool,** etc.




```
public class Kalkylator
{
    //Deklaration av instansvariabel
    private int tal = 0;

    public void EnAnnanMetod()
    {
        int resultat = 0;  //deklration och intiering

        resultat = ÖkaTalMedEtt(8);

    }

    //tillägg är en metod paramter, och har typen int
    //Metoder som anropar denna metod måste skicka ett värde av int-typ
    public int ÖkaTalMedEtt(int tillägg)
    {
        // Deklaration of en lokal variabel,skapas och initieras vid varje metod anrop.
        int resultat = 0; //initiering

        resultat = tal + tillägg; //Tilldelnngs sats.
        return resultat;
    } //variabeln resultat dör här
}
```

4. **Deklaration av retur-typen av metoder som inte är void.**

Metoder som inte deklareras som **void**, måste returnera ett värde av någon typ. Typen måste bestämmas vid definition av metoden. I det föregående exemplet, har metoden **ÖkaTalMedEtt** en retur typ som är **int**. Metoden returnerar ett **int** värde (resultat).

```
public int ÖkaTalMedEtt(int tillägg)
{
    // Deklaration of en lokal variabel,skapas och initieras vid varje metod anrop.
    int resultat = 0; //initiering

    resultat = tal + tillägg; //Tilldelnngs sats.
    return resultat;
} //variabeln resultat dör här
```

10. Sammanfattning

Variabler måste deklareras innan de används. Vid deklarationen bestäms typen och ett namn för en variabel. Variabler som deklareras som fält i en klass nås av alla metoder i klassen och de initieras eller nollställas d.v.s. får ett startvärde av kompilatorn om man inte gör det explicit i klassen. Variablerna förlorar sina minnesplatser när objektet har städats av skräppsamlaren (Garbage Collector).

Variabler som deklareras inne i en metod kallas för lokala variabler. Variablerna lever endast till dess metoden är under exekvering. När exekvering når den sista raden i metoden (slut klammarparentesen), gäller inte variabeln längre. Lokala variabler måste explicit tilldelas ett initialt värde. När metoden anropas nästa gång, allokeras de lokala nya minnesplatser och initieras på nytt.

Det är många inbyggda värdebaserade datatyper att välja mellan, men använd följande tumregel:

- double för reella tal (samt tal som kan innehålla både heltal och tal med bråkdel)
- int för heltal,
- bool för logiska värden,
- char för enstaka tecken,
- decimal för data där man vill ha större noggrannhet som t ex valuta och finansiella beräkningar.

Dokumentet fortsätter i Del 2 där vi kommer att lära oss skillnaderna mellan värdetyper och referenstyper. Till dess, kom ihåg att referenstypen string används för text och den mäktiga inbyggda strukturen **DateTime** passar utmärkt för tid och datum värden.

Lycka till.
Farid Naisan,
Kursansvarig och lärare