Objektorienterad programmering
Föreläsning 4

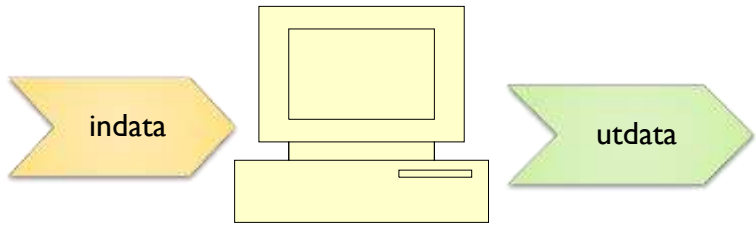
Mahmud Al Hakim
mahmud@dynamicos.se
www.webbacademy.se

Agenda

- Introduktion till objektorientering
- Klasser och Objekt
- Instansvariabler
- Metoder
- Introduktion till UML, klassdiagram
- Datatyper (enkla typer och referenstyper)
- Tilldelning och typomvandling
- Mer om operatorer

Den klassiska bilden av ett datorprogram

Programmets uppgift är att transformera ett dataflöde.



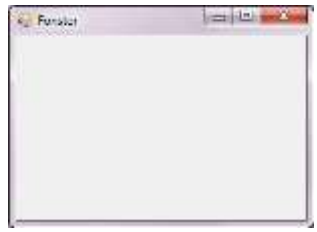
Det funktionsorienterade synsättet

Objektorientering

- Vid objektorienterad programmering bygger man upp sina program av ett antal väl avgränsade enheter som kallas **objekt**.
- Ett datorprogram är en modell av den verklighet programmet skall samverka med.
- De enskilda enheterna (objekten) är modeller av verkliga eller tänkta ting i programmets omgivning.
- Ett datorprograms uppgift är att manipulera objekten.
- I programmet använder man s.k. **klasser** för att avbilda och beskriva objektens egenskaper.

Det objektorienterade synsättet

```
Class Bil
{
    …
}
```




```
Class Fönster : Form
{
    …
}
```

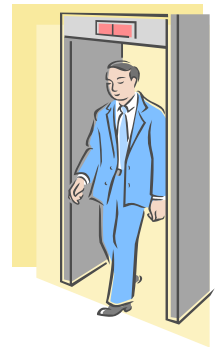
Modeller av verkliga eller tänkta föremål

Objekt

- Varje objekt har en unik identitet.
- Man kommer åt objekt via namn, antingen direkt eller via referenser.
- Varje objekt beskrivs av sina **tillståndsvariabler** och **operationer (metoder)**.
- Tillståndsvariabler används för att hålla reda på objektets tillstånd eller status.
- Varje objekt har en egen, unik uppsättning tillståndsvariabler som brukar gömmas inne i objektet.
- Tillståndsvariablerna kan förändras av objektet självt men är oåtkomliga utifrån.
- Detta kallas **inkapsling** (information hiding).

Exempel – Objektet hissen

- Objektet beskriver en verklig hiss.
- Dess tillstånd kan beskrivas av två **tillståndsvariabler**:
  - **Riktning**: kan ha något av värdena, *stilla*, *uppåt* och *neråt*
  - **Våning**: kan innehålla ett *heltal* som anger vilken våning hissen är på.



Tillståndsvariabler kallas även **instansvariabler**

Operationer (metoder)

- För ett objekt beskriver man också de **operationer** som man kan utföra på objektet.
- För objektet *hissen* kan det t.ex. finnas metoderna
  - KörTill: används för att få hissen att köra till en viss våning.
  - Stanna: används för att stoppa hissen.
  - VilkenVåning: används för att ta reda på var hissen f.n. befinner sig.
- En sådan operation kallas för en **metod** eller **instansmetod**.

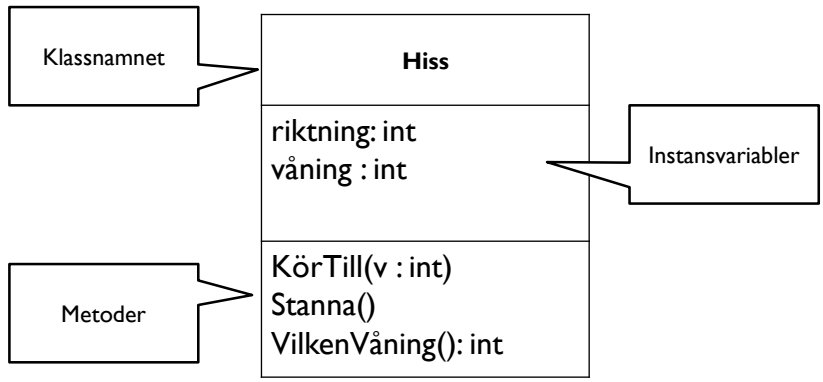
Klass

- En klass är en mall, eller mönster, som beskriver hur objekt med samma uppbyggnad ser ut.
- En klass är en *beskrivning* av objekt.
- En klass ska du se som en byggritning.
- Objekt skapas från klasser.
- Det måste alltså finnas en klass innan man kan skapa ett objekt.
- Man kan ha olika klasser som beskriver olika sorters objekt.

UML (Unified Modeling Language)

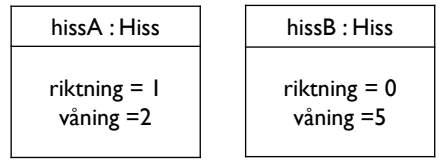
- UML är ett objektorienterat generellt språk för modellering av alla typer av system.
- Genom att skapa en modell av systemet som skall konstrueras blir det enklare att förstå och bygga det.
- UML innehåller regler för hur man visuellt presenterar klasser, objekt och andra ting som har med objektorientering att göra.
- Tips:
  http://www.uml.org

UML Klassdiagram



UML Objektdiagram

- Ett objekt som tillhör en viss klass är en instans av klassen.
- Det kan finnas flera objekt som tillhör en viss klass.



Två instanser av klassen Hiss

Deklarera klasser i C#

```
class Hiss
{
    // instansvariabler
    int riktning; // -1 neråt, 0 stilla, 1 uppåt
    int våning;

    // metoder
    public void KörTill(int v)
    {
        // ...
    }
    public void Stanna()
    {
        // ...
    }
    public int VilkenVåning()
    {
        // ...
    }

}
```


En klass som beskriver en person

- Vi ska nu konstruera en klass som beskriver en person.
- Några vanliga instansvariabler är ålder, vikt och längd.
- Exempel på operationer som en person kan ha/göra
  SättÅlder, FyllerÅr, SättVikt och SättLängd.
- Med hjälp av klassen kan vi även beräkna en persons BMI (Body Mass Index).

Klassen Person

```
class Person
{
    // Instansvariabler
    int ålder;
    double vikt; // kg
    double längd; // m

    // Metoder
    public void SättÅlder(int age)
    {
        ålder = age;
    }
    public void SättVikt(double weight)
    {
        vikt = weight;
    }
    public void SättLängd(double lenght)
    {
        längd = lenght;
    }
    public void FyllerÅr()
    {
        ålder = ålder+1;
    }
}
```


Ett objekt av klassen Person

```
static void Main(string[] args)
{
    Person p = new Person();
    p.SättÅlder(40);
    p.SättVikt(90);
    p.SättLängd(1.7);
}
```

p är ett objekt av klassen Person

Med hjälp av metoderna ändrar vi objektets **tillståndsvariabler**

En metod som beräknar BMI

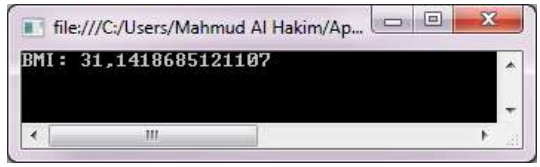
```
public double GetBMI()
{
    double bmi;
    bmi = vikt / (längd * längd);
    return bmi;
}
```

$$\text{BMI} = \frac{m}{l^2} = \frac{vikten\ (\text{kg})}{längden\ \times\ längden\ (\text{m}^2)} \left[\frac{kg}{m^2}\right]$$

http://sv.wikipedia.org/wiki/Body_Mass_Index

Visa objektets BMI

```
static void Main(string[] args)
{
    Person p = new Person();
    p.SättÅlder(40);
    p.SättVikt(90);
    p.SättLängd(1.7);
    Console.WriteLine("BMI: "+p.GetBMI());
    Console.ReadKey();
}
```



Gränsvärden för BMI definierade av WHO.

Viktklasser	BMI
Undervikt	mindre än 18,5
Normalvikt	18,5–24,9
Övervikt	25,00–29,9
Fetma grad 1	30,0-34,9
Fetma grad 2	35,0-39,9
Fetma grad 3	mer än 40

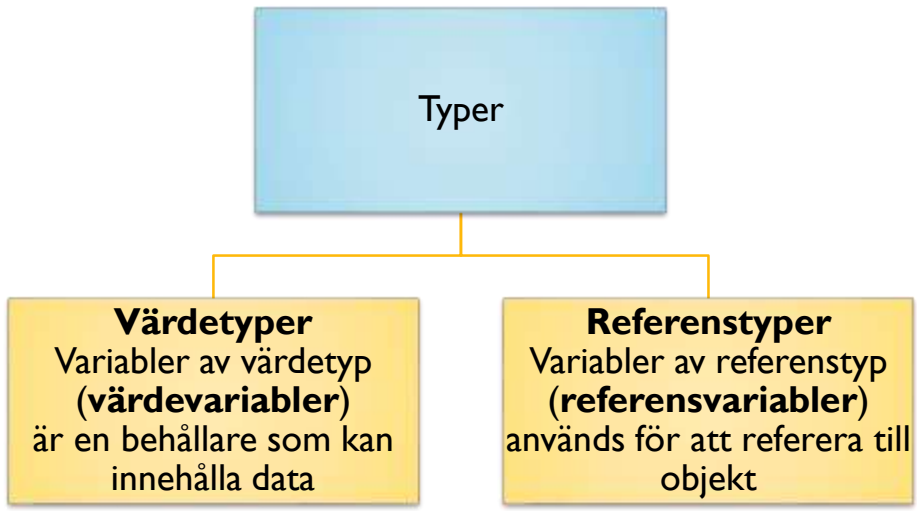
Egenskaper (Properties)

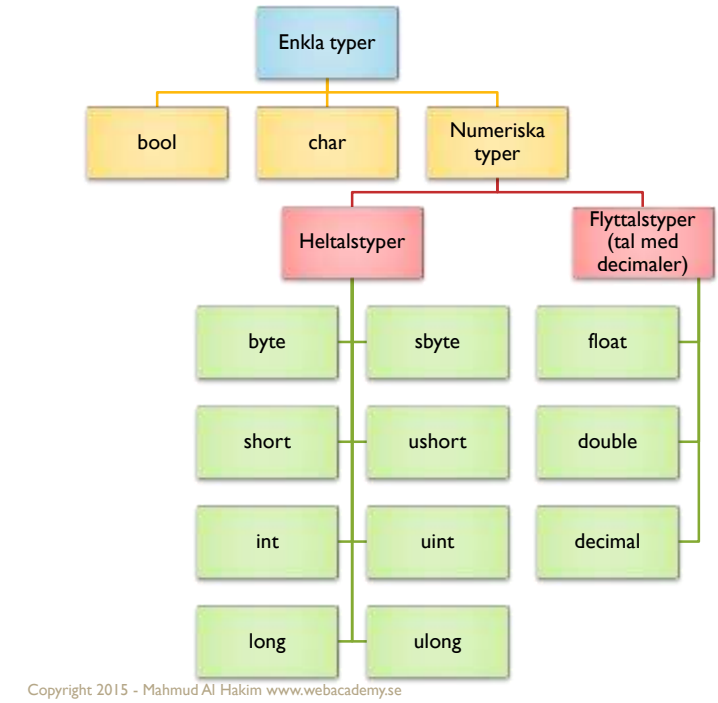
- Egenskaper gör att man kan avläsa och ändra instansvariabler.
- Egenskaper kan ha en avläsare (**get**) och en ändrare (**set**).
- Ändraren har en underförstådd variabel med namnet **value**.
- När man deklarerar en egenskap som direkt motsvarar en instansvariabel brukar man låta egenskapen ha samma namn som instansvariabeln, fast med **stor begynnelsebokstav.**


```
class Person
{
    // Instansvariabler
    int ålder;
    double vikt;  // kg
    double längd; // m

    // Egenskaper
    public int Ålder
    {
        get { return ålder; }
        set {
            if (value > 0) ålder = value;
            else Console.WriteLine("Felaktig ålder!");
        }
    }
    public double Vikt
    {
        get { return vikt; }
        set {
            if (value > 0) vikt = value;
            else Console.WriteLine("Felaktig vikt!");
        }
    }

    // Skriv egenskapen Längd själv
```

```
        static void Main(string[] args)
        {
            Person p = new Person();
            p.Ålder = 40;
            p.Vikt = 85;
            Console.WriteLine("Ålder: "+ p.Ålder);
            Console.WriteLine("Vikt: " + p.Vikt);
            Console.ReadKey();
        }
    }
```


Typer





Integral Types Table (Från C# Reference)

Type	Range	Size
sbyte	-128 to 127	Signed 8-bit integer
byte	0 to 255	Unsigned 8-bit integer
short	-32,768 to 32,767	Signed 16-bit integer
ushort	0 to 65,535	Unsigned 16-bit integer
int	-2,147,483,648 to 2,147,483,647	Signed 32-bit integer
uint	0 to 4,294,967,295	Unsigned 32-bit integer
long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Signed 64-bit integer
ulong	0 to 18,446,744,073,709,551,615	Unsigned 64-bit integer

http://msdn.microsoft.com/en-us/library/exx3b86w.aspx

Flyttalstyper (tal med decimaler)

Typ	Minsta till Största värde	Storlek	Noggrannhet
float	$-3{,}4 \times 10^{38}$ till $+3{,}4 \times 10^{38}$	32 bitar	7 siffror
double	$-1{,}7 \times 10^{308}$ till $+1{,}7 \times 10^{308}$	64 bitar	15-16 siffror
decimal	$-7{,}9 \times 10^{28}$ till $7{,}9 \times 10^{28}$	128 bitar	28-29 siffror

Typen **decimal** är lite speciell. Till skillnad mot float och double lagrar den talen ”exakt”.

Den kräver fler bitar och talområdet är inte så stort som för flyttalstyperna.

Används i första hand för applikationer där man räknar pengar.

MinValue och MaxValue

- För alla typer gäller att man kan använda egenskaperna MinValue och MaxValue för att få reda på det minsta resp. största tal som kan lagras i en variabel av typen.

```
static void Main(string[] args)
{
    Console.WriteLine("Typen int minsta värde: " + int.MinValue);
    Console.WriteLine("Typen int största värde: " + int.MaxValue);
    Console.WriteLine("Typen float minsta värde: " + float.MinValue);
    Console.WriteLine("Typen float största värde: " + float.MaxValue);
    Console.ReadKey();
}
```

Literaler

- Man behöver ofta ange konstanta värden i ett uttryck tex.

  x + 12.6

  i / 10
- Sådana konstanta värden kallas **literaler**.
- I uttrycken ovan är 10 en **heltalsliteral** och 12.6 en **reell literal**.

Tilldelning och typomvandling

variabelnamn = uttryck;

- Uttrycket till höger beräknas först.
- Dess värde placeras sedan i variabeln till vänster.
- Uttrycket måste ha samma typ som variabeln eller automatiskt kunna omvandlas till denna typ (implicit typomvandling) t.ex.

```
int i;
double d;
d = i;  // OK, ofarlig typomvandling, int ryms i double
i = d;  // FEL, farlig typomvandling!!!
```

Explicit typomvandling (cast)

- Om man **vill** göra ”farliga” typomvandlingar måste man använda en explicit typomvandling s.k. cast.
- Då skriver man inom parentes framför uttrycket i högerledet vilken typ man vill att uttryckets värde skall omvandlas till, t.ex.

  i = (int) d; // OK, explicit typomvandling
- OBS! När man omvandlar från reell typ till heltalstyp ”kapas” decimalerna. Vi får inte avrundning.

Sammansatta tilldelningar

x **+=** y	Samma som x = x + y
x **-=** y	Samma som x = x - y
x ***=** y	Samma som x = x * y
x **/=** y	Samma som x = x / y
x **%=** y	Samma som x = x % y

Typen char

- För att lagra ett tecken i datorn brukar man oftast använda en grupp som består av 8 bitar (en byte).
- Med hjälp av 8 bitar kan man bilda 256 olika tecken.
- I C# används Unicode 16 bitar (s.k. BMP*), vilket kan koda 65536 tecken.
- Variabler av standardtypen char är i C# 16 bitar långa.
- Teckenvärden (teckenliteraler) skrivs omgivna med apostrofer t.ex.

```
char ch1 = 'A';
```

* BMP: Basic Multilingual Plane

Referenstyper

- Klasstyper i C# är referenstyper.
- När man deklarerar en variabel av en referenstyp så får man inte ett objekt, utan **en referens**.
- Denna referens kan sedan användas för att komma åt objektet, t.ex.

```
Hiss h1;
```

- h1 är en referensvariabel som har förmågan att referera till hissar.
- h1 har typen ”**referens till Hiss**”.
- För att få h1 att referera till ett objekt måste vi först skapa ett objekt (skapa en instans av klassen Hiss) t.ex.

```
h1 = new Hiss();
```

Identifierare

- Namn på saker och ting i ett program brukar med ett finare ord kallas för *identifierare*.
- En identifierare för vara godtyckligt lång och kan bestå av bokstäver, siffror och understrykningstecken.
- **En identifierare får inte börja med en siffra.**
- Stora och små bokstäver betraktas olika.
- Man får inte använda de reserverade orden som namn Tips, en lista på alla keywords finns här: http://msdn.microsoft.com/en-us/library/x53a06bb.aspx
- Klassnamn och metodnamn brukar inledas med en stor bokstav.
- Variabler brukar inledas med en liten bokstav.

Numeriska uttryck

- I program förekommer ofta uttryck där man beräknar numeriska tal.
- Sådana utryck kallas ”**numeriska uttryck**”.
- I dessa kan man använda de vanliga matematiska operationer
  - Addition +
  - Subtraktion –
  - Multiplikation *
  - Division /
- Den vanligaste formen har två operander t.ex.

$$x + y$$

Heltalsdivision och operatorn %

- Vid division, om någon av operanderna är ett reellt tal då sker vanlig division och resultatet blir ett reellt tal.
- Om båda operanderna är av heltalstyp t.ex. int så utförs s.k. **heltalsdivision**.
- Det betyder att man ser hur många gånger den högra operanden ”går i” den vänstra. T.ex.

```
int x = 14;
int y = 5;
```

- Uttrycket **x/y** ger resultatet 2 (5 går 2 gånger i 14).
- För att få fram resten vid heltalsdivision används **operatorn %**
- Uttrycket **x%y** ger resultatet 4

Prioritet

- * / och % har högst prioritet.
- + och - har lägst prioritet.
- I ett komplicerat uttryck bestämmer de olika operatorernas prioriteter i vilken ordning uttrycket beräknas.
- Operatorer med högre prioritet beräknas före operatorer med lägre prioritet.
- Om två operatorer har samma prioritet sker beräkningen från vänster till höger. T.ex.

-2+4/2*3 = 4

- Man kan använda parenteser för att styra beräkningsordningen. T.ex. (-2+4)/2*3 = 3

Öknings- och minskningsoperatorer

- Följande satser kan med hjälp av operatorerna ++ och -- skrivas enklare

  i = i + 1; → i++;

  k = k - 1; → k--;

- Öknings- och minskningsoperatorer finns i två varianter

  1. Postfix t.ex. i++;
  2. Prefix t.ex. ++i;

Operatorerna ++ och -- som postfix

```
static void Main(string[] args)
{
    int x = 4;
    int y = 7;
    int z;
    z = x++ * y--; //postfix, z får värdet 4*7=28
    Console.WriteLine("z=" + z);
    Console.WriteLine("x=" + x); // x är nu 5
    Console.WriteLine("y=" + y); // y är nu 6
    Console.ReadKey();
}
```

Operatorerna ++ och -- som prefix

```
static void Main(string[] args)
{
    int x = 4;
    int y = 7;
    int z;
    z = ++x * --y; //prefix, z får värdet 5*6=30
    Console.WriteLine("z=" + z);
    Console.WriteLine("x=" + x); // x är nu 5
    Console.WriteLine("y=" + y); // y är nu 6
    Console.ReadKey();
}
```


Typen bool

- Typen **bool** används för att beskriva sanningsvärden.
- Har enbart två tillåtna värden, **false** och **true**.
- En instansvariabel av typen bool som inte explicit initieras får automatiskt värdet false.
- Typen bool används mest i samband med **jämförelser**.
- Förekommer bl.a. i if-satser och while-satser.
- Ett jämförelseuttryck har typen bool och bildas med hjälp av *jämförelseoperatorer*.

Jämförelseoperatorer

==	Lika med
!=	Icke lika med
<	Mindre än
<=	Mindre än eller lika med
>	Större än
>=	Större än eller lika med

Logiska operatorer

- Man kan bilda s.k. logiska uttryck med hjälp av operatorerna

  && utför operationen och

  || utför operationen eller

  ! utför operationen icke
- A && B sant om både A och B är sanna
- A || B sant om minst en av A eller B är sanna
- !A sant om A är falsk
- Ex. (temp>20 && temp <30)