Föreläsning 1

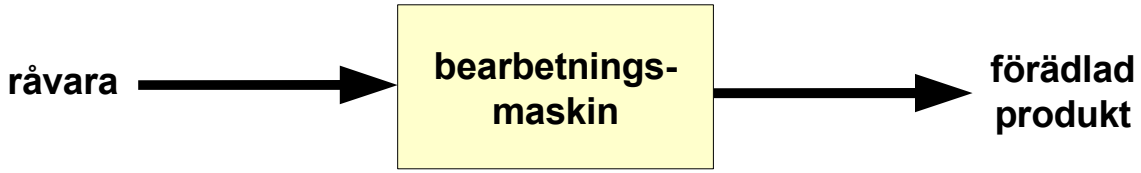
Introduktion till programutveckling

Varför ha kännedom om datateknik och programmering?

- datorerna har blivit en förutsättning för det västerländska industrisamhällets existens och dess framtida tekniska utveckling
- datorer finns som komponenter i alla typer av tekniska system
- man bör ha kännedom av den teknik man använder
- många avancerade tekniska tillämpningsprogram är programmerbara och för att utnyttja dessa till fullo behövs grundläggande kunskaper i programmering
- man bör känna till de möjligheter som programmering av datorer erbjuder och de svårigheter som hör till
- den problemlösningsmetodik som används är tillämpningsbar inom många andra områden
- . . .

Vad är en dator?

Ett sätt (av många) är att betrakta en dator som en bearbetningsmaskin, vilken i likhet med andra bearbetningsmaskiner tar en råvara och från denna producerar en förädlad produkt.



För en dator utgörs både råvaran och produkten av data.

Data vs. information

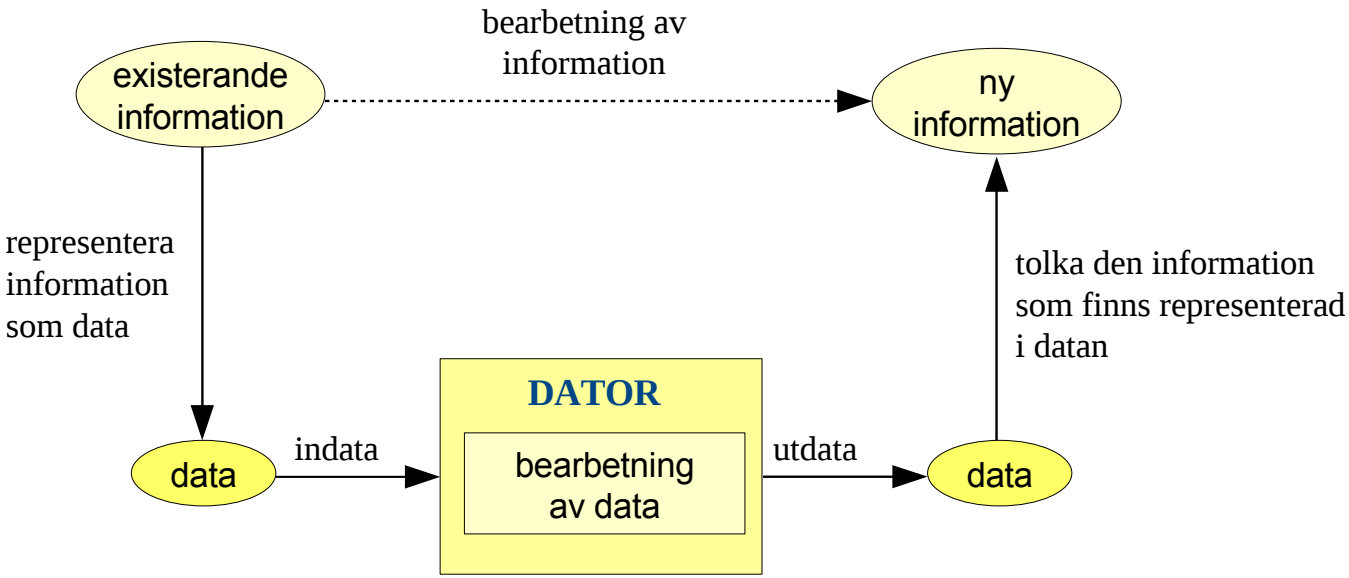
När vi bearbetar data i en dator är det inte indatavärdena och utdatavärdena i sig själva som är det intressanta, utan den *information* som representeras av dessa datavärden.

Information kan definieras som den innebörd en mottagare lägger in i givna data. Informationen uppstår först när mottagen data tolkas av mottagaren och därmed får en viss innebörd.

data + tolkning = information

Data vs. information

En dator är således en bearbetningsmaskin för information. För att kunna bearbeta informationen måste denna ges till datorn i form av data. Datorn kan sedan, på ett eller annat sätt, bearbeta dessa data. Som resultat av bearbetningen erhålles någon form av utdata som sedan kan tolkas för att utvinna ny information.



Mer om data

Data är kodad information som kan förekomma i många olika former, t.ex tal, texter, bilder, ljud och ljus.

data = kodad information

Samma information kan presenteras på många olika dataformat, och samma data kan med olika tolkning representera olika information.

Exempel:

Hur kan heltalet 6 kodas?

Med 10-talssystemet:	6
Med romerska siffror:	VI
Med "streck":	𝍸 /
Med binära talsystemet:	110_2
Med bokstäver:	"sex"

Mer om data

Exempel:

Hur kan 123 tolkas?

- som tecknet ’1’ följt av tecknet ’2’ följt av tecknet ’3’

  Rätt kombination till portkoden

- som strängen "123"

  Namnet på en pub

- som heltalet 123 i 10-talssystemet (123_{10})

  $1*10^2 + 2*10^1 + 3*10^0$

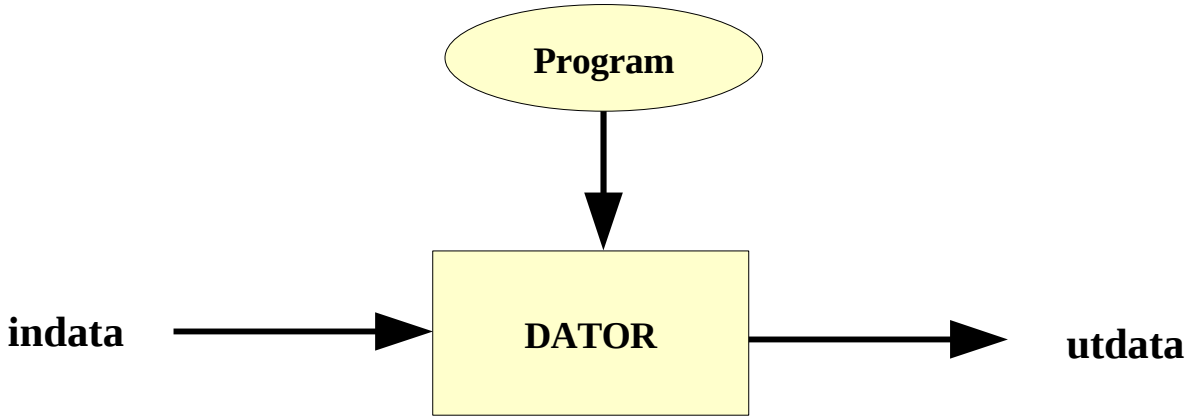
- som heltalet 123 i 8-talssystemet (123_8)

  $1*8^2 + 2*8^1 + 3*8^0 = 83_{10}$

Det interna dataformatet i en dator är binära tal.

Datorprogram

Den grundläggande idén, som gör datorn så användbar och flexibel, är att dess arbete styrs via ett datorprogram. Datorprogrammen kan bytas ut samt varieras och utformas allt efter de behov och krav som den aktuella tillämpningen ställer. En dator har således inte en specificerad arbetsuppgift, utan är ett *generellt verktyg* för att bearbeta information som kan utnyttjas för nästan alla användningsområden.



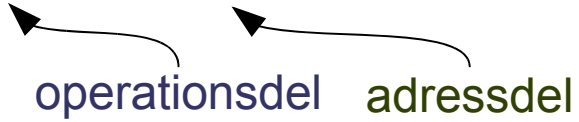
Datorprogram

Internt i datorn är ett program på ett format som kallas *maskinkod*. Maskinkoden utgörs av binära tal:

0010000000111000

0011000000111001

0101000000111010



Maskinkoden består av en operationsdel och en adressdel. Operationsdelen anger vad som skall göras och adressdelen anger var den information finns som berörs av operationen.

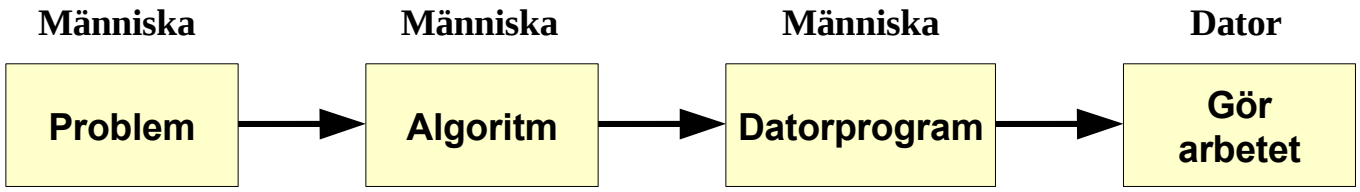
Att skriva program direkt i maskinkod är allt annan än människovänligt, varför särskilda *programspråk* har utvecklats.

Vad är ett datorprogram?

Ett dataprogram är en *beskrivning* av hur ett problem skall lösas, uttryckt på sådant sätt att datorn kan *förstå* och *utföra* de olika stegen i beskrivningen.

En beskrivning av hur ett visst problem skall lösas kallas för en *algoritm*.

Först när man har en algoritm kan man börja skriva sitt program med hjälp av ett programmeringsspråk.



Ett problem kan oftast lösas på *många* olika sätt, dvs det kan finnas många algoritmer för ett och samma problem.

Således finns det i allmänhet inte endast en rätt lösning till ett programmeringsproblem, utan många. En rätt lösning kan dock vara en dålig lösning, eftersom det vanligtvis finns (som vi kommer att se senare) fler krav på ett datorprogram än att det enbart skall producera rätt resultat.

Algoritmer

En algoritm beskrivs i termer av en *ändlig* följd av elementära steg eller instruktioner.

Alla algoritmer kan uttryckas med hjälp av följande enkla styrkonstruktioner:

- sekvens (följd)
- selektion (val)
- iteration (upprepning)
- hopp (skall normalt inte användas, ger svårbegripliga algoritmer)

Algoritmer

Då stegen i en algoritm utförs - på ett noggrant sätt - resulterar detta i att den arbetsuppgift som beskrivs av algoritmen blir verkställd, dvs en algoritm måste *terminera*.

För att kunna utföra arbetsuppgiften som beskrivs av en algoritm måste man:

- kunna förstå varje steg i algoritmen
- kunna utföra de instruktioner som stegen beskriver.

Stegen i en algoritm måste således vara *otvetydiga* och *exekverbara*.

Algoritmer

I instruktionsboken för ett frysskåp finns under rubriken ”Om skåpet inte fungerar tillfredsställande” följande algoritm:

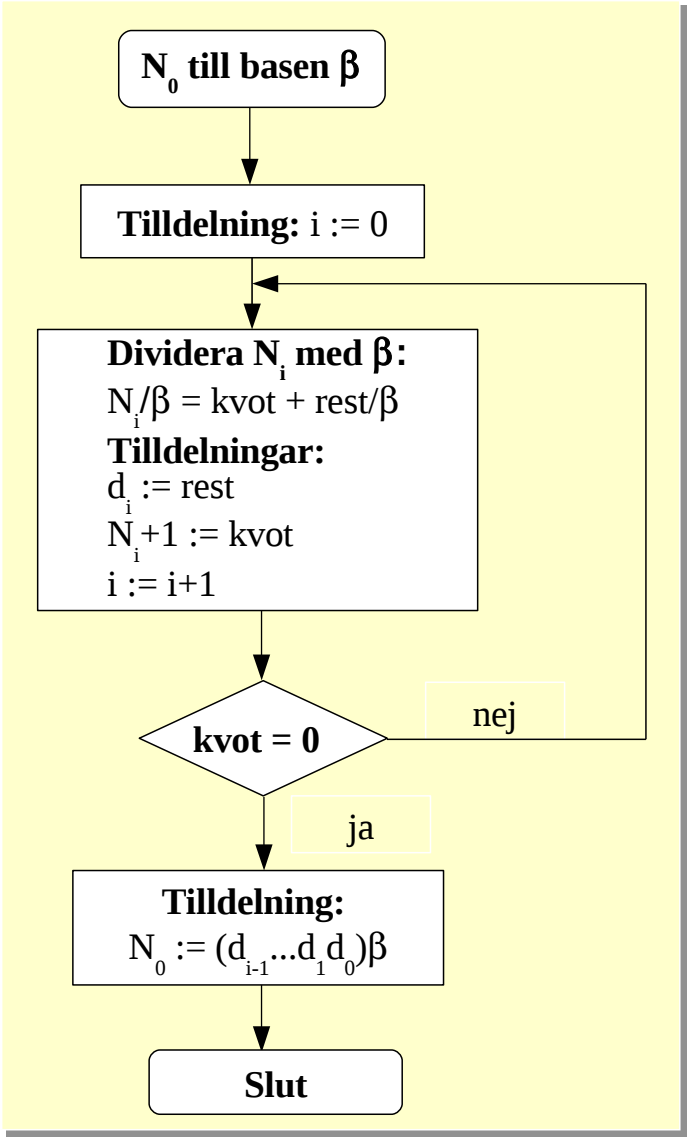
Undersök följande innan Ni begär service:

- Att stickproppen sitter i ordentligt.
- Att säkringen är hel.
- Att det inte är strömavbrott.
- Att alla manöverprogram är rätt inställda.
- Att dörren är ordentligt stängd.
- Att skåpet inte står för nära en värmekälla.
- Att inte ett tjockt frost/islager bildats.

Om kompressorn gör upprepade startförsök utan resultat, stäng av skåpet i 20 min och försök sedan på nytt ett par gånger.

Algoritmer

Flödesdiagrammet beskriver en algoritm för att omvandla ett decimalt heltal N_0 till basen β.



Algoritmer

En algoritm är *oberoende* av vilket språk den är beskriven i.

Ett datorprogram är en algoritm som är beskriven på sådant sätt att stegen i algoritmen kan *förstås* och *utföras* av en dator.

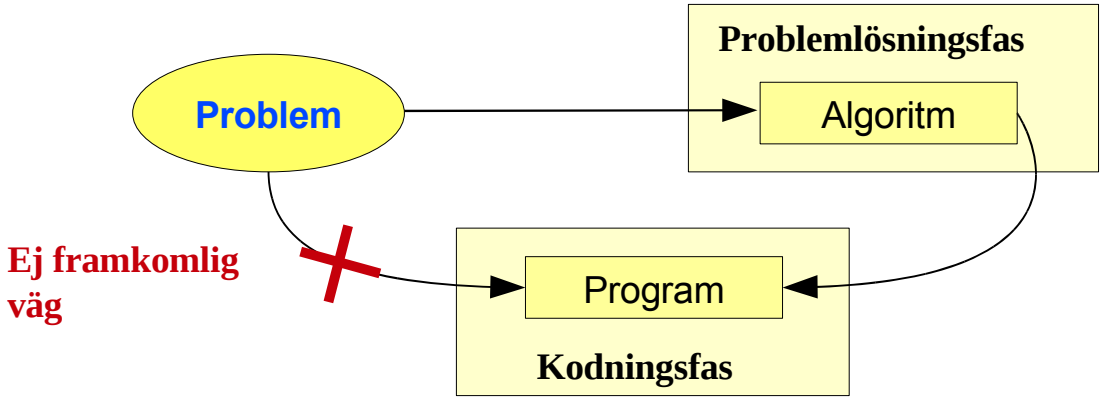
För att kunna skriva ett datorprogram för att lösa ett givet problem måste man ha tillgång till en algoritm som ger en lösning till problemet *innan* man kan skriva datorprogrammet!

programkonstruktion => algoritmkonstruktion => problemlösning

När vi utvecklar algoritmer använder vi oss oftast av *pseudokod*, vilket är en informell blandning av ett "riktigt" programspråk och ett mänskligt språk som svenska eller engelska.

Algoritmer

Om man utvecklar algoritmen direkt som ett datorprogram kommer man att dränkas i detaljer. Lösningen blir ostrukturerad, bristfällig och oftast felaktig.



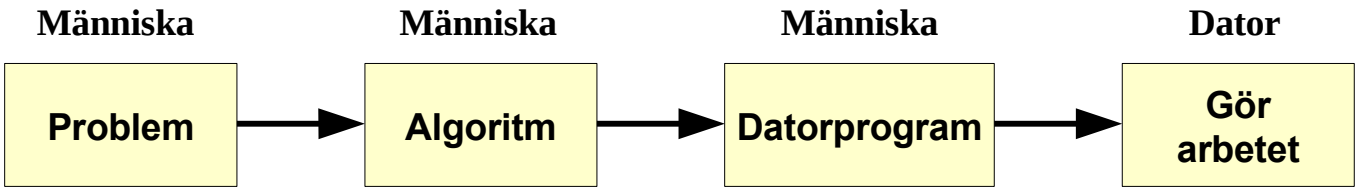
”Koda direkt”-metoden leder i bästa fall till program som är:

- svåra att förstå
- svåra att validera
- svåra att lokalisera fel i
- svåra att korrigera fel i
- svåra att anpassa till nya utvidgningar.

Vid utveckling av ett större program är det troligast att man överhuvudtaget inte lyckas skriva ett fungerande program!

Algoritmer

- Först när man har en algoritm kan man börja skriva sitt program med hjälp av ett programmeringsspråk.
- Ett programspråk tillhandahåller de styrkonstruktioner som erfordras för att representera en algoritm så att den kan utföras av en dator.
- Datorer gör endast det de blir instruerade att göra - det är programmerarens ansvar att algoritmen är riktig och kodas i programspråket på ett korrekt sätt.



Algoritmer

- Lika viktigt som att programmet är begripligt för datorn, lika viktigt är det att programmet är begripligt för människan.
- Ett program är inte en isolerad och oföränderlig enhet – programmet ingår vanligtvis i ett större system och är i allmänhet i behov av återkommande underhåll och modifieringar, t.ex på grund av förändringar i det överordnade systemet (t.ex. orsakad av nya användarkrav, ny hårdvara, ny organisation eller ny lagstiftning).

Råd på vägen

Tänk först, koda sedan!

Ju tidigare du börjar koda, ju längre tid kommer det att ta innan du har ett fungerande program!

Vad är programmering?

Programmering kan definieras som *samtliga arbetssteg som behövs för att kunna lösa ett problem med hjälp av en dator*.

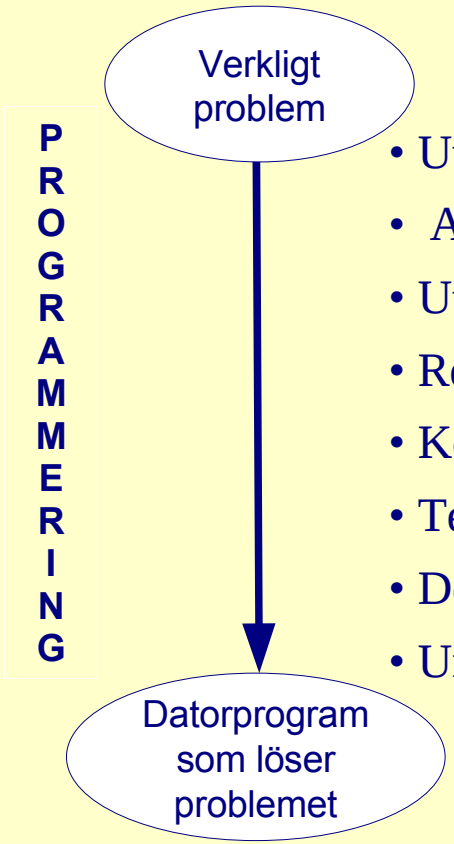
Att kunna programmera är således inte enbart att behärska ett visst programspråk, utan framförallt att känna till metoder för att strukturera och lösa problem så att en dator kan användas som hjälpmedel.

Programmering handlar i mycket stor utsträckning om *problemlösning* och *metodkännedom*.

För att bli framgångsrik måste programmeraren ha en disciplinerad och strukturerad arbetsmetodik.

Vad är programmering?

Programmeringsarbetet kan indelas i följande faser:



- Utarbeta en exakt definition av problemet
- Analysera problemet
- Utarbeta en lösningsskiss/modell till problemet
- Representera lösningen i en lämplig algoritm
- Kodning i ett lämpligt programspråk
- Testning och validering av programmet
- Dokumentation av programmet
- Underhåll av programmet

Vad är ett programspråk?

Att skriva ett program är att instruera datorn vad den skall göra. Problemet med en dator är att den kräver att få instruktioner på ett speciellt sätt (s.k. maskinkod), vilket skiljer sig ifrån det sätt som människor sinsemellan kommunicerar.

När vi vet vad vi vill få datorn att göra, vill vi alltså översätta från någonting vi kan förstå till någonting som datorn kan förstå.

En tanke skulle vara att vi uttryckte det vi ville göra, dvs programmerade, i mänskligt språk. Men problemet med det mänskliga språket är att det är omfångsrikt, mångtydigt och inte strikt definierat.

Den kompromissen man gjort mellan mänskligt språk och maskinkod är att skapa särskilda programmeringsspråk, som är strikt definierade men fortfarande liknar något vi människor är vana att förstå och formulera.

I kursen kommer vi att använda programmeringsspråket Java.

Varför Java?

Java är ett modernt programspråk med flera tilltalande egenskaper:

- stödjer strukturerad programmering
- är ett objektorienterat språk, vilket underlättar utveckling av stora programsystem
- är plattformsoberoende
- tillhandahåller verktyg för att skapa grafiska användargränssnitt
- är utvecklat med tanke på Internet-användningar
- har möjligheter till att skriva parallella program
- tillhandahåller ett omfattande klassbibliotek, med färdigskrivna programmoduler
- integrerar ny teknologi med nya klassbibliotek
- tillgång till bra kompilatorer som finns kostnadsfritt
- relativt lätt att lära sig

Förberedelser inför kodning: Analys

Problem:

Skriv ett program som läser två heltal och skriver ut summan av talen.

Analys:

Indata: De två heltalen som skall adderas.

Utdata: Summan av de inlästa talen.

Exempel på körning:

Ange första talet: **5**
Ange andra talet: **10**
Summan av talen är 15

Förberedelser inför kodning: Design

Design:

Algoritm:

1. Skriv texten *”Ange första talet: ”*.
2. Läs *tal1*.
3. Skriv texten *”Ange andra talet: ”*.
4. Läs *tal2*.
5. Addera *tal1* och *tal2* och spara resultatet i *summa*.
6. Skriv texten *”Summan av talen är ”*.
7. Skriv ut *summa*.

Datarepresentation:

tal1, *tal2* och *summa* är heltal (som i Java avbildas med hjälp av datatypen ***int***).

Innan implementationen

För att kunna skriva ett program som implementerar algoritmen ovan måste vi veta hur man i det aktuella programspråket:

- avbildar objekten i algoritmen som dataobjekt
- skriver ut text
- läser värden till heltalsvariabler
- adderar två heltalsvariabler
- lagrar ett värde i en heltalsvariabel
- skriver ut text
- skriver ut värdet av heltalsvariabler

Detta skall vi förhoppningsvis lära oss innan föreläsningen är slut.

Strukturen hos ett Javaprogram

Ett program i Java består av ett antal samverkande klasser, som kommunicerar med varandra via meddelanden för att lösa uppgiften.

Programmet har en huvudklassen, vilken innehåller en `main`- metod som är själva startpunkten för programmet.

Mall för "enkla program"

```
public class Klassnamn {
    public static void main (String[] args) {
        deklarationer och satser
    }//main
}//Klassnamn
```

Ett första Javaprogram

```
public class Hello {
    public static void main (String[] args) {
        System.out.println("Hello world!");
        System.out.print("This is a message from the computer.");
    } // main
} // Hello
```

Programmet skriver ut texten

Hello world!

This is a message from the computer.

i datorns kommandofönster

Ett första Javaprogram

```
public class Hello {
   public static void main (String[] args) {
      System.out.println("Hello world!");
      System.out.print("This is a message from the computer.");
   } // main
} // Hello
```

Namnet vi valt på huvudklassen (programmet) är `Hello`.

`main`-metoden består av två *satser*:

```
System.out.println("Hello world!");
System.out.print("This is a message from the computer.");
```

Varje sats avslutas med ett *semikolon* (;).

Båda satserna är *anrop* till klassmetoder i klassen `System`, dvs klassen `Hello` samverkar med klassen `System`.

När man gör ett anrop av en metod brukar man säga att man *skickar meddelande*. Klassen `Hello` skickar alltså meddelanden till klassen `System`.

Något om klassen `System`

Klassen `System` kan (något förenklat) sägas vara en uppsättning programenheter som någon redan utvecklat. Klassen `System` innehåller bland annat ett antal så kallade *klassmetoder* med vilka andra program kan kommunicera för att få saker utförda.

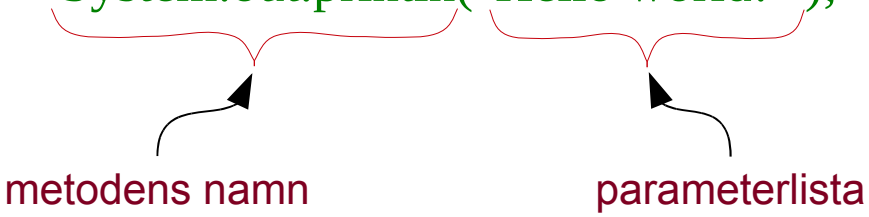
Metoderna

```
System.out.println(det_som _skall_skrivas_ut)
System.out.print(det_som_skall_skrivas_ut)
```

används för att få utskrifter i kommandofönstret.

En metod består av ett namn och en parameterlista.

```
System.out.println("Hello world! ");
```



Något om klassen `System`

Skillnaden mellan metoderna `print` och `println` är att metoden `println` automatiskt skriver ut ett radslutstecken, vilket betyder att *nästa* utskrift hamnar på en ny rad.

I anropet

```
System.out.println("Hello world!");
```

är det en textsträng vi vill skriva ut. För att ange att det rör sig om en textsträng måste textsträngen omges av *citationstecken*.

Klassen `System` finns i Javas *standardbibliotek* (API:n) i *paketet* `java.lang`.

Kompilering och exekvering

Vårt program

```
public class Hello {
    public static void main (String[] args) {
        System.out.println("Hello world! ");
        System.out.print("This is a message from the computer.");
    } // main
} // Hello
```

måste lagras på en textfil med namnet Hello.java, dvs namnet på klassen samt suffixet .java.

I filen Hello.java finns nu programmet i form av *källkod*.

För att kunna *exekvera* (köra) programmet måste källkoden först översättas till ett format som förstås av datorn. Denna översättning görs med hjälp av en *kompilator*.

Kompilering och exekvering

För att kompileringen skall lyckas måste programmet vara *syntaktiskt korrekt*, dvs följa de språkregler som finns i Java. Annars uppstår *kompileringsfel*, pga att kompilatorn inte förstår vad som programmeraren menar.

När kompileringen av källkoden lyckas, skapas en ny fil Hello.class. Denna fil innehåller programmet i ett format som kallas *Javabytekod* och detta format förstås av datorn.

Används kommandofönstret kompileras programmet med kommandot

```
javac Hello.java
```

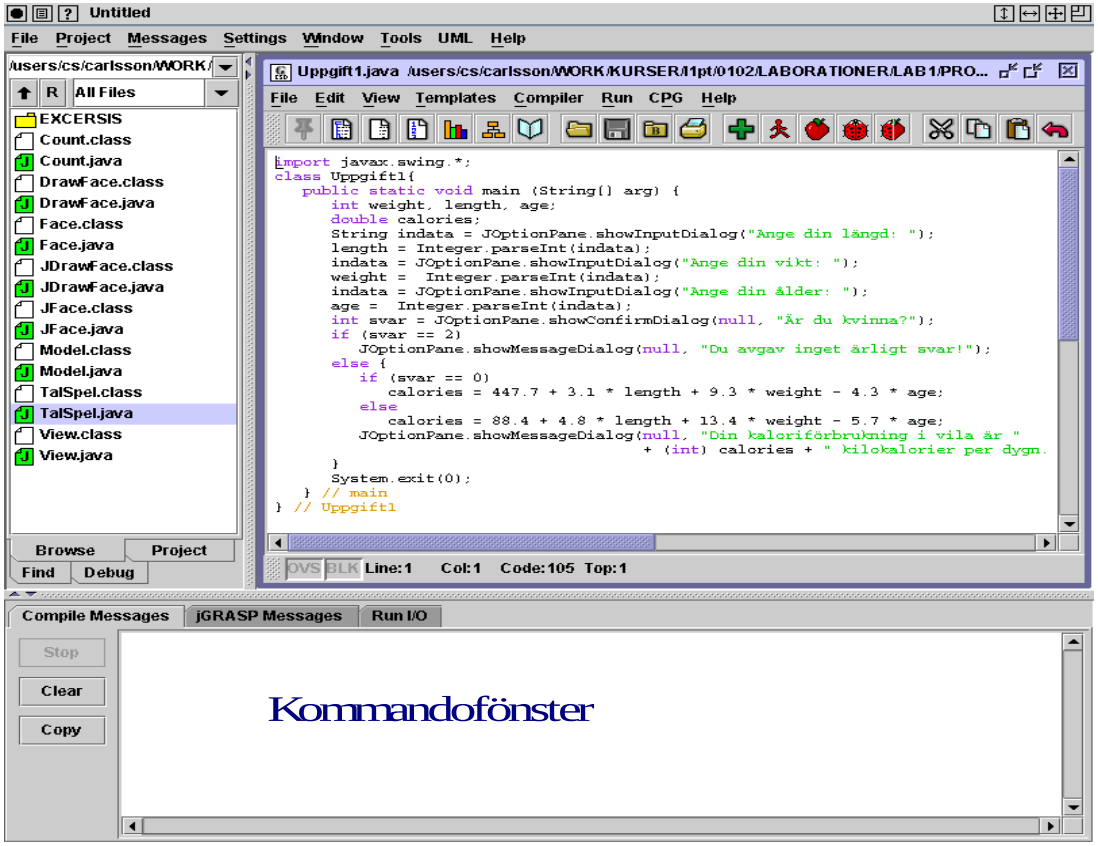
och exekveras med kommandot

```
java Hello
```

I kursen kommer vi att använda en speciell texteditor, JGrasp, i från vilken man kan både kompilera och exekvera programmet.

JGrasp

JGrasp är en texteditor i vilken man får olika former av stöd vid skrivandet av sitt program, och från vilken man kan kompilera och exekvera programmet.



Användning av dialogrutor för utskrift

I Java finns ett paket som heter `Swing`, som innehåller standardklasser för att skapa grafiska användargränssnitt.

I `Swing` finns klassen `JOptionPane` som innehåller metoder för att skapa dialogrutor för in- och utmatning. För utmatning har klassen `JOptionPane` bl.a metoden `showMessageDialog`.


```
import javax.swing.*;
public class Hello2 {
    /* Detta program ger en hälsning från datorn */
    public static void main (String[] args) {
        JOptionPane.showMessageDialog(null,"Hello world! \n" +
                                            "This is a message from the computer.");
    } // main
} // Hello2
```

Användning av dialogrutor för utskrift

```
import javax.swing.*;
public class Hello2 {
   /* Detta program ger en hälsning från datorn */
   public static void main (String[] args) {
      JOptionPane.showMessageDialog(null,"Hello world! \n" +
                                          "This is a message from the computer.");
   } // main
} // Hello2
```

Kommentarer:

För att få tillgång till metoden `JOptionPane.showMessageDialog` måste paketet `Swing` *importeras*, vilket görs med satsen

```
import javax.swing.*;
```

Specialtecknet '/n' anger radslutstecken.

Operatorn + används (i denna kontext) för att slå ihop två textsträngar.

Note: Klassen `System` som användes i förra programmet finns i ett paket som heter `java.lang`, detta paket behöver dock inte importeras eftersom detta görs automatiskt.

Användning av dialogrutor för utskrift

Ett program kan innehålla flera dialogrutor för utskrift.

```
/* Detta program skriver ut två meddelanden i var sin dialogruta */
import javax.swing.*;
public class Hello3 {
    public static void main (String[] args) {
        JOptionPane.showMessageDialog(null,"Hello world! \n");
        JOptionPane.showMessageDialog(null, "This is a message from the computer.");
    } // main
} // Hello3
```

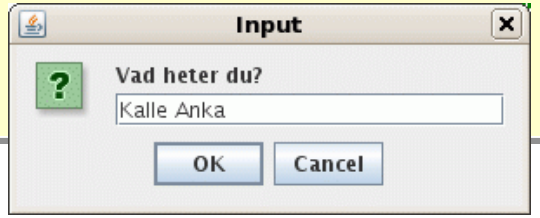
Först visas meddelanderutan

när man trycker på knappen OK kommer nästa meddelanderuta upp

Dialogrutor för inmatning

För inmatning har klassen `JOptionPane` bl.a metoden `showInputDialog`.

```
import javax.swing.*;
public class Greeting {
    public static void main (String[] arg) {
        String name = JOptionPane.showInputDialog("Vad heter du?");
        String greeting = "Välkommen " + name;
        System.out.println(greeting);
    } //main
} // Greeting
```



Om användaren, som i exemplet ovan, skriver Kalle Anka i dialogrutan kommer texten

Välkommen Kalle Anka

att skrivas ut i kommandofönstret när användaren trycker på OK-knappen.

Dialogrutor för inmatning

```
import javax.swing.*;
public class Greeting {
    public static void main (String[] arg) {
        String name = JOptionPane.showInputDialog("Vad heter du?");
        String greeting = "Välkommen " + name;
        System.out.println(greeting);
    } //main
} // Greeting
```

Kommentarer:

Metoden `JOptionPane.showInputDialog` returnerar en sträng (allt som skrivs in via tangentbordet är strängar!).

Denna sträng måste tas om hand och lagras i en variabel av klassen `String`.

Klassen `String` används i Java för att avbilda strängar.

Textvariabler

- Textvariabler avbildas i Java med hjälp av standardklassen `String`.
- För att tilldela en variabel ett värde används *tilldelningsoperatorn* =.
- För att slå samman två texter finns för klassen `String` operatorn +.

Exempel:

När nedanstående satser utförs

```
String texten;
texten = "Hej";
texten = texten + " Kalle";
```

kommer variabeln `texten` att refererar till ett objekt som innehåller texten "Hej Kalle".

Klassen `String` kommer att behandlas mer utförligt senare i kursen.

Dialogrutor för inmatning av text

Istället för att som i förra exemplet göra utskriften i kommandofönstret kan utskriften ske i en dialogruta:

```
import javax.swing.*;
public class Greeting2 {
    public static void main (String[] arg) {
        String name = JOptionPane.showInputDialog("Vad heter du?");
        String greeting = "Välkommen " + name;
        JOptionPane.showMessageDialog(null, greeting);
    } //main
} // Greeting2
```

När programmet körs visas först dialogrutan för inmatning

Användaren skriver in efterfrågad indata och trycker på OK-knappen, varvid följande dialogruta visas

Inbyggda primitiva typer i Java

I Java finns något som kallas för *primitiva typer* (eller *enkla typer*) som används för att avbilda enkla slag av objekt och som används som byggstenar för att konstruera mera komplexa objekt.

byte	för att avbilda heltal	8 bits
short	för att avbilda heltal	16 bits
int	för att avbilda heltal	32 bits
long	för att avbilda heltal	64 bits
float	för att avbilda reella tal	32 bits
double	för att avbilda reella tal	64 bits
boolean	för att avbilda logiska värden	
char	för att avbilda tecken	

För att avbilda heltal kommer vi enbart att behandla **`int`** och för att avbilda reella tal kommer vi enbart att behandla **`double`**.

Vad är en variabel?

I ett datorprogram används *variabler* för att lagra olika typer av data.

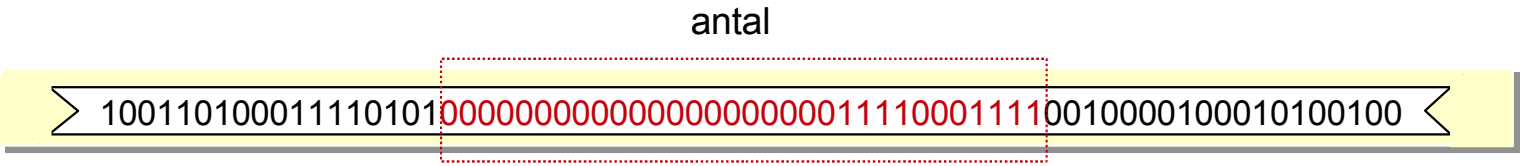
I Java finns olika slag av variabler och de variabler som används för att lagra primitiva datatyper kallas *enkla variabler*.

En variabel kan ses som *en namngiven behållare i vilken man kan lagra ett värde av en viss typ*.

Observera!!

I matematiken betyder begreppet variabel något annat.

Variabels namn kopplas till ett visst minnesutrymme i datorns primärminne där variabelns värde lagras i form av *bits*.



Deklarationer av variabler

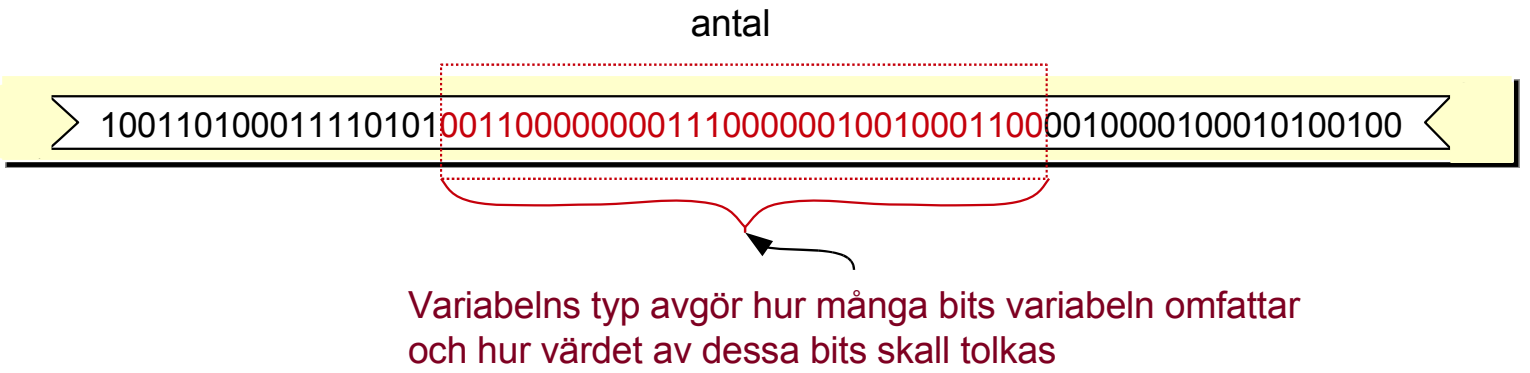
En variabel måste deklareras innan den används i programmet.

Variabeldeklarationer har följande utseende:

```
int antal;
```

Storleken (antalet bits) på minnesutrymmet som associeras med en variabel beror på variabelns *datatyp*.

Bitmönstret i minnesutrymmet som är associerat med en variabel bestämmer tillsammans med variabelns datatyp vilket *värde* variabeln har.



Deklarationer av variabler

Deklarationerna

```
int antal;
double vikt, produktPris;
```

innebär att tre variabler skapas.

Värdet av dessa variabler är *odefinierat* (eftersom värdet bestäms av det bitmönster som *råkar* ligga i minnesutrymmet). Värdet av en variabel är odefinierat tills variabeln explicit har tilldelats ett värde i programmet.

När *tilldelningssatserna*

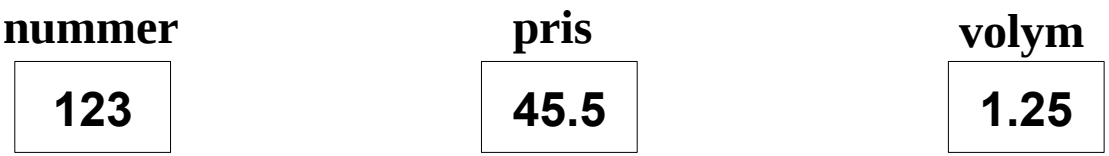
```
antal = 10;
vikt = 1.87;
produktPris = 24.75;
```

har utförts har respektive variabel tilldelas värden:

Deklarationer av variabler

En variabel kan tilldelas ett värde direkt i deklarationssatsen:

```
int nummer = 123;
double pris = 45.5, volym = 1.25;
```



En variabel deklareras *exakt en gång*, dvs varje variabel måste ha ett unikt namn. Deklareras samma variabler flera gånger erhålls ett kompileringsfel.

Exempel:

```
int bredd = 123;
double bredd = 45.5;
```

En felutskrift fås från kompilatorn

```
"bredd is already defined"
```

vid satsen

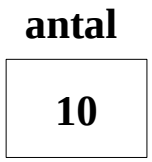
```
double bredd = 45.5;
```

Deklarationer av variabler

- Värdet av en variabel kan när som helst läsas av.
- En variabel kan när som helst tilldelas ett nytt värde.

Antag att vi gjort följande variabeldeklaration

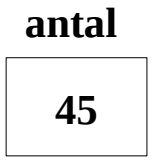
```
int antal = 10;
```



Utförs nu tilldelningssatsen

```
antal = antal + 35; //antal tilldelas värdet av antal + 35
```

förändras värdet på variabeln `antal`



Observera de två olika betydelserna variabeln `antal` har i tilldelningssatsen

```
antal = antal + 35;
```

minnesutrymmet för variabeln

innehållet i minnesutrymmet för variabeln

Operationer

Att endast lagra data i variabler är ganska ointressant. Syftet med ett datorprogram är att från någon form av indata producera utdata. Utdatan är, i en eller annan mening, en förädlad form av indatan.

För att från indatan kunna producera utdata, måste vi kunna göra *beräkningar* på de värden som lagras i variablerna i programmet.

De primitiva datatyperna har ett antal fördefinierade *operationer*, som används för att utföra beräkningar.

Gemensamt för dessa operationer är att de tar två värden (*operander*) och returnerar ett nytt värde.

Med hjälp av operationerna kan man bygga upp komplicerade *uttryck*.

Operationer på datatypen `int`

Notation	Betydelse	Resultatets datatyp
+	addition	**int**
-	subtraktion	**int**
*	multiplikation	**int**
/	heltalsdivision	**int**
%	rest vid heltalsdivision	**int**
>	större än	**boolean**
<	mindre än	**boolean**
>=	större eller lika med	**boolean**
<=	mindre eller lika med	**boolean**
==	lika med	**boolean**
!=	inte lika med	**boolean**

Motsvarande för **`byte`**, **`short`** och **`long`**

Omslagsklasser

Till var och en av de primitiva typerna finns en omslagsklass, som innehåller information om datatypen samt en del användbara metoder.

Omslagsklasserna heter: `Integer`, `Double`, `Character`, `Boolean`, . . .

Omslagsklassen `Integer` innehåller bl.a följande konstanter och metoder:

```
public static final int MAX_VALUE
public static final int MIN_VALUE
public static String toString(int i)
public static int parseInt(String s)
public static Integer valueOf(int i)
```

Kommentar:

public anger att entiteten är nåbar utanför klassen

static anger att entiteten är en klassentitet

Dessa begrepp kommer att förklaras utförligt senare.

Omslagsklasser

Exempel:

Anropet Integer.parseInt("1234") returnerar heltalet 1234

Anropet Integer.toString(5678) returnerar strängen "5678"

Satserna

```
System.out.println("Största heltal: " + Integer.MAX_VALUE);
System.out.println("Minsta heltal: " + Integer.MIN_VALUE);
```

ger utskriften:

```
Största heltal: 2147483647
Minsta heltal: -2147483648
```

Det färdiga programmet

```
/* Programmet läser in och adderar två heltal, samt skriver ut resultatet. */
import javax.swing.*;
public class Addera {
    public static void main (String[] arg) {
        String indata = JOptionPane.showInputDialog("Ange första talet");
        int tal1 = Integer.parseInt(indata);
        indata = JOptionPane.showInputDialog("Ange andra talet");
        int tal2 = Integer.parseInt(indata);
        int summa = tal1 + tal2;
        JOptionPane.showMessageDialog(null, "Summan av talen är " + summa);
    } //main
} // Addera
```