Föreläsning 1

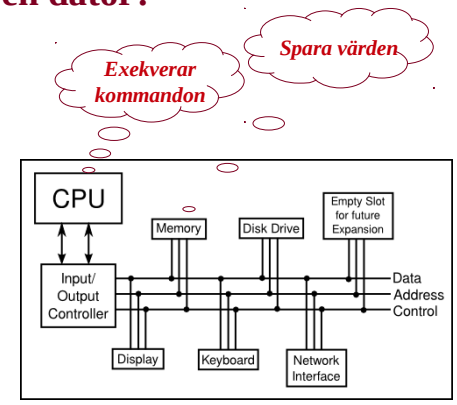
Introduktion till programutveckling

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 (i en bil kan uppemot 100 datorer finnas).
- Man bör ha kännedom av den teknik man använder.
- Många avancerade tekniska tillämpningsprogram är programmerbara, 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?

- von Neumann modell
- Transformera indata till utdata
- Exekvera kommando
  - Kommando:
    uppdrag att ändra minne*
- Program:
  - stor sekvens av kommando
  - utfört ett efetr ett av CPUn

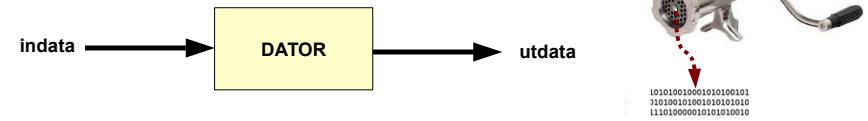


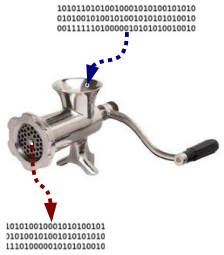
*Programmet finns också i minnet men ändras inte.

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

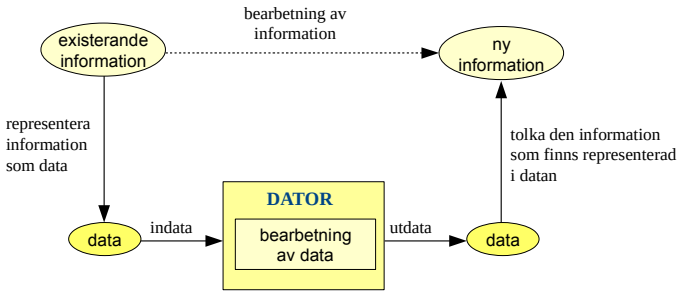
När vi bearbetar data i en dator är det inte indatavärdena och utdatavärdena i sig själva som är det intressanta, utan den ***information*** som representeras av dessa datavärden.

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	(eller 6_{10} för att särskilja mellan olika talsystem)
Med romerska siffror:	VI	
Med "streck":	𝍸 /	
Med binära talsystemet:	110	(eller 110_2 för att särskilja mellan olika talsystem)
Med bokstäver:	"sex"	

Mer om data

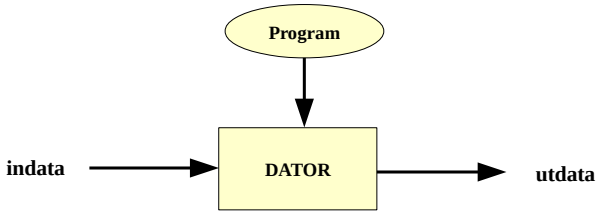
Exempel: Hur kan 123 tolkas?

- som tecknet ’1’ följt av tecknet ’2’ följt av tecknet ’3’
  t.ex. rätt kombination till portkoden
- som strängen "123"
  t.ex. 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, d.v.s. sekvenser av 0 och 1.

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 och kan utnyttjas för nästan alla användningsområden.



Datorprogram

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

	Tänkbar betydelse operationsdelen:
0010000000111000	*läs in innehållet på adressen till register A*
0011000000111001	*addera innehållet på adressen till innehållet i register A*
0101000000111010	*spara register A på adressen*

operationsdel adressdel

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

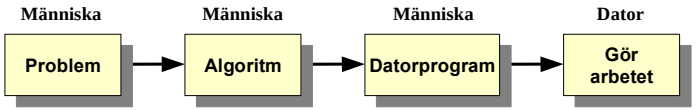
Att skriva program direkt i maskinkod är allt annan än människovänligt, varför särskilda ***programspråk*** har utvecklats.

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 är en ordnad, ändlig följd av elementära och entydiga operationer/instruktioner som löser en klass av uppgifter.

Ordnad: Den följd i vilken operationerna utförs är bestämd.

Ändlig: Algoritmen kommer alltid till ett slut.

Elementära: Varje operation gör *en sak* (på den aktuella abstraktionsnivån).

Entydiga: Det är aldrig någon tvekan om vad en viss operation innebär, och betydelsen ändras inte under tillämpningen av algoritmen.

Löser: Algoritmen utmynnar i ett resultat som relaterar till den givna uppgiften. Resultatet kan vara framgång eller misslyckande, men det finns alltid ett resultat och resultatet är alltid giltigt.

Klass av uppgifter: Algoritmen kan användas för att lösa varje praktisk situation där ett problem av den aktuella formen föreligger.

Algoritmer

Då operationerna i en algoritm ***exekveras*** - 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

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

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

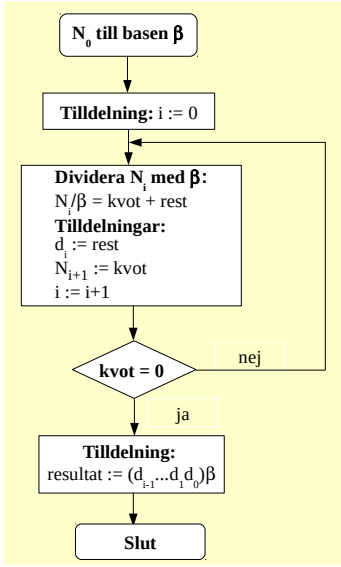
Undersök följande innan Ni begär service:

1. Att stickproppen sitter i ordentligt.
2. Att säkringen är hel.
3. Att det inte är strömavbrott.
4. Att alla manöverprogram är rätt inställda.
5. Att dörren är ordentligt stängd.
6. Att skåpet inte står för nära en värmekälla.
7. 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

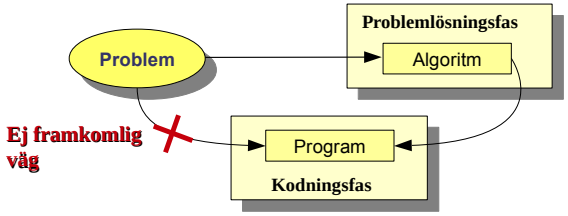
- En algoritm är *oberoende av språket* i vilken den är beskriven.
- Ett datorprogram är en algoritm som är beskriven på sådant sätt att stegen i algoritmen kan *förstås* och *utföras* av en dator.
- För att kunna skriva ett datorprogram för att lösa ett givet problem måste man ha tillgång till en algoritm 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.



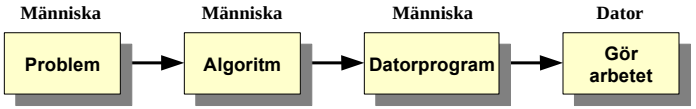
”Koda direkt”-metoden leder i bästa fall till program som är:

- 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 algoritmen 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 (orsakad av nya användarkrav, ny hårdvara, ny organisation eller ny lagstiftning).

Råd på vägen

Tänk först, koda sedan!

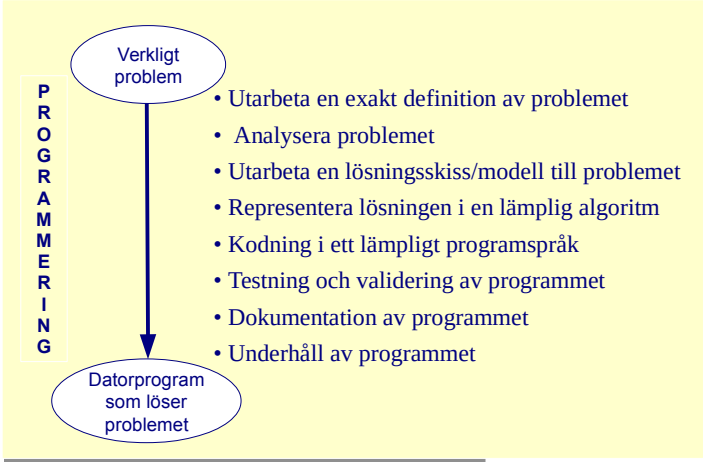
Ju tidigare du börjar koda, ju längre tid kommer det att ta innan du har ett fungerande program!

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:



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.

- Igår sköt jag en hare med gevär på 100 meter.
- Ett gevär på 100 meter, det var då ingen liten bössa.
- Nä, nä. Med en bössa sköt jag en hare på 100 meter.
- En hare på 100 meter. Det var då ingen dålig stek du fick.
- Fattar du inte. På 100 meter sköt jag en hare med gevär.
- En hare med gevär? Då var det i alla fall tur att du sköt först.

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.

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 värdet av heltalsvariabler

Detta skall vi förhoppningsvis lära oss innan föreläsningen är slut.

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.

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 ***huvudklass***, 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! ");
```

metodens namn — parameterlista

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 ett ***paket*** med namnet 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 ***kompileras***, d.v.s. översättas till ett format som förstås av datorn. Kompileringen görs med hjälp av en ***kompilator***.

Kompilering och exekvering

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

Filen Hello.java innehåller programmet i form av ***källkod***.

För att kunna ***exekvera*** (köra) programmet måste man ***kompilera*** källkoden.

Kompileringen görs av en ***kompilatorn***.

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

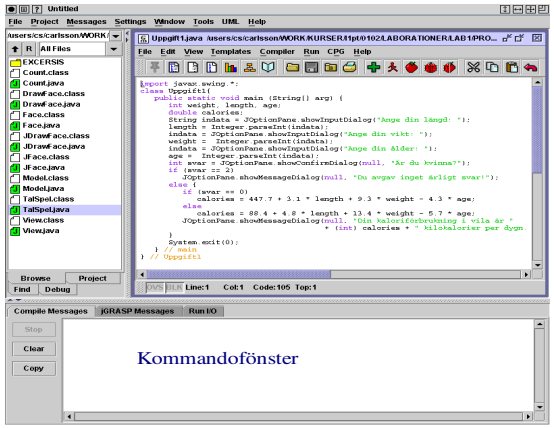
och exekveras med kommandot

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.JOptionPane;
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.JOptionPane;
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:

- Klassen JOptionPane finns i paketet javax.swing och och måste ***importeras*** till programmet, vilket görs med satsen

  ```
  import javax.swing.JOptionPane;
  ```

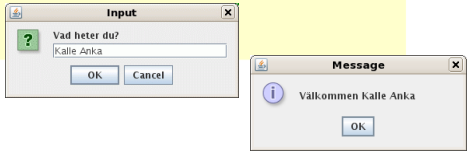
- '\n' är ett ***specialtecken*** som anger radslut (***radslutstecken***).
- Operatorn + används (i denna kontext) för att slå ihop två textsträngar.

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.

Dialogrutor för inmatning

För inmatning har klassen JOptionPane bl.a metoden showInputDialog.

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



Kommentarer:

Metoden showInputDialog returnerar en textsträng (allt som skrivs in via tangentbordet är text!). Denna sträng måste tas om hand och lagras i en variabel av klassen String, som används i Java för att avbilda textsträngar.

Inläsningen från showInputDialog aktiveras när användaren trycker OK-knappen.

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.

Inbyggda primitiva typer i Java

I Java finns 8 olika ***enkla typer*** (eller ***primitiva typer***) som används för att avbilda enkla slag av objekt och som används som byggstenar för att konstruera mera komplexa objekt.

Datatyp	Användning	Storlek
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	16 bits
char	för att avbilda tecken	för att avbilda tecken

Numeriska datatyper i Java

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

Datatyp	Storlek	Min. värde	Max. värde
byte	8 bits	-128	127
short	16 bits	-32768	32767
int	32 bits	-2147483647	2147483647
long	64 bits	-9223372036854775808	9223372036854775807
float	32 bits	ca +/-1.4 E-45 7 siffrors noggrannhet	ca +/3.4 E+38 7 siffrors noggrannhet
double	64 bits	ca +/- 4.9 E-324 15 siffrors noggrannhet	ca +/-1.8 E+308 15 siffrors noggrannhet

Vad är en variabel?

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

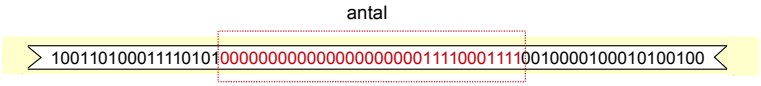
I Java finns olika slag av variabler och de variabler som används för att lagra enkla datatyper kallas ***enkla variabler***.

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

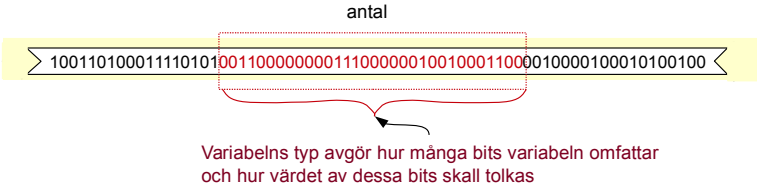
En variabel måste ***deklareras*** innan den används i programmet.

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.

Dessa variabler har ***odefinierade värden*** (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***

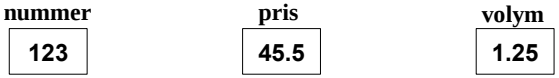
```
antal = 10;
vikt = 1.87;
produktPris = 24.75;
```

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;
```

Felaktig deklaration

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.

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

Operationer på datatypen int

Notation	Betydelse	Resultatets datatyp
a + b	addition	**int**
a - b	subtraktion	**int**
a * b	multiplikation	**int**
a / b	heltalsdivision	**int**
a % b	modulus (rest vid heltalsdivision)	**int**
a > b	större än	**boolean**
a < b	mindre än	**boolean**
a >= b	större eller lika med	**boolean**
a <= b	mindre eller lika med	**boolean**
a == b	lika med	**boolean**
a != b	inte lika med	**boolean**
+a	samma som a	**int**
-a	negationen av a	**int**

Motsvarande operationer finns för **byte**, **short** och **long**.

Heltalsdivision och rest vid heltalsdivision

Utryck	Dividend	Divisor	Kvot	Rest	
26 / 6 26 % 6	26	6	4	2	26 = 4 * 6 + 2
100 / 40 100 % 40	100	40	2	20	100 = 2 * 40 + 20
2 / 4 2 % 4	2	4	0	2	2 = 0 * 4 + 2
-5 / 2 -5 % 2	-5	2	-2	-1	-5 = -2 * 2 + -1
7 / -3 7 % -3	7	-3	-2	1	7 = -2 * -3 + 1

Omslagsklasser

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

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

Omslagsklassen Integer innehåller bl.a följande konstanter och metoder:

static final int MAX_VALUE	det största värdet som kan lagras i en **int**
static final int MIN_VALUE	det minsta värdet som kan lagras i en **int**
static String toString(**int** n)	ger heltalet n som en sträng
static int parseInt(String str)	ger strängen str som en **int**

Kommentar:

static anger att entiteten är en klassentitet

final anger att entiteten inte kan förändra sitt värde

Dessa begrepp kommer att förklaras utförligt senare.

Omslagsklasser

Exempel:

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

Anropet Integer.parseInt("abc") ger NumberFormatException

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.JOptionPane;
public class AddTwoIntegers {
   public static void main (String[] arg) {
      String input = JOptionPane.showInputDialog("Ange första talet");
      int number1 = Integer.parseInt(input);
      input = JOptionPane.showInputDialog("Ange andra talet");
      int number2 = Integer.parseInt(input);
      int sum = number1 + number2;
      JOptionPane.showMessageDialog(null, "Summan av talen är " + sum);
   } //main
} // AddTwoIntegers
```