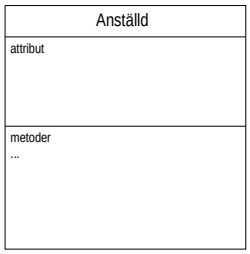
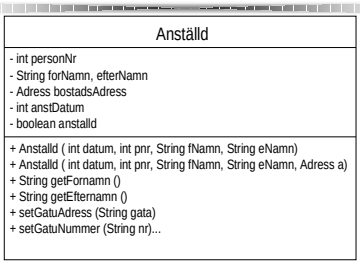
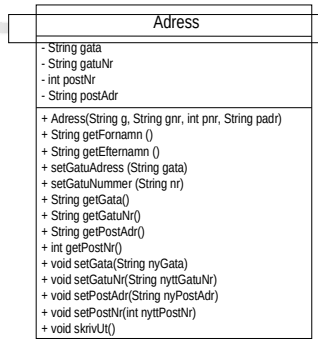
Anställd



Anställd



Adress



Anställd - skal

```
public class Anstalld
{
  //Attribut som normalt är private
  private int personNr;  //Personnummer
  ...

  //Konstruktor(er)
  public Anstalld(int datum, int pnr, String fNamn, String eNamn)
  {
    anstDatum = datum;
        ...
  }//konstruktorn Anstalld

  //Metoder
  public int getPersonNr()
  {
    return personNr;
  }//metod getPersonNr
  …

}//Anstalld
```


Anställd - attribut

```
public class Anstalld
{
  private int anstDatum;  //När personen blev anställd
  private int personNr;  //Personnummer
  private Adress bostadsAdress;
  private String forNamn,efterNamn;  //för- & efternamn
  private boolean anstalld;  //Anställd eller slutat
  …




}
```


Anställd-konstruktorer

```
public Anstalld(int datum, int pnr, String fNamn, String eNamn)
{
  anstDatum = datum;
  personNr = pnr;
  forNamn = fNamn;
  efterNamn = eNamn;
  anstalld = true;
  bostadsAdress = new Adress("","",0,""); //tomt adressobjekt
}//konstruktor

public Anstalld(int datum, int pnr, String fNamn, String eNamn, Adress a)
{
  anstDatum = datum;
  personNr = pnr;
  forNamn = fNamn;
  efterNamn = eNamn;
  anstalld = true;
  bostadsAdress = a;
}//konstruktor
```

Anställd - metoder

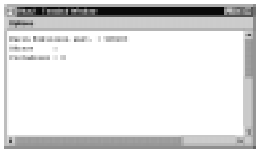
```
//Servicemetoder för attributen
  public String getFornamn()
  {
    return forNamn;
  }//metod getFornamn

  public String getEfternamn()
  {
    return efterNamn;
  }//metod getEfternamn
...
```


`skrivUt` - hjälprutin

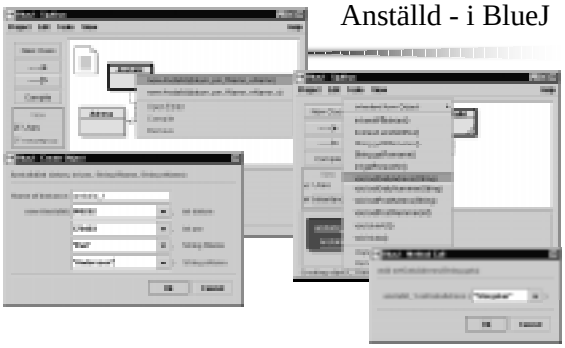
```
public void skrivUt()
{
  System.out.print(forNamn+" "+getEfterNamn());
  if (anställdNu())
  {
    System.out.println(" anst. : "+anstPåbörjad());
    bostadsAdress.skrivUt();
  }
  else
    System.out.println(" har slutat");
}//metod skrivUt
```


`skrivUt` i `Adress`

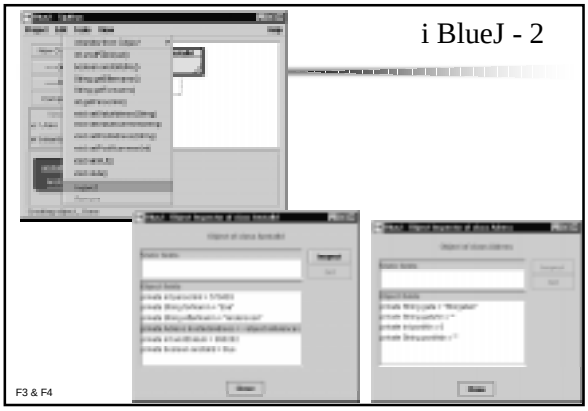
```
// Enkel utskriftsrutin för kontroll
public void skrivUt()
{
   System.out.println("Adress     : " + gata + " " + gatuNr);
   System.out.println("Postadress : " + postNr + " " + postAdr);

}//metod skrivUt
```



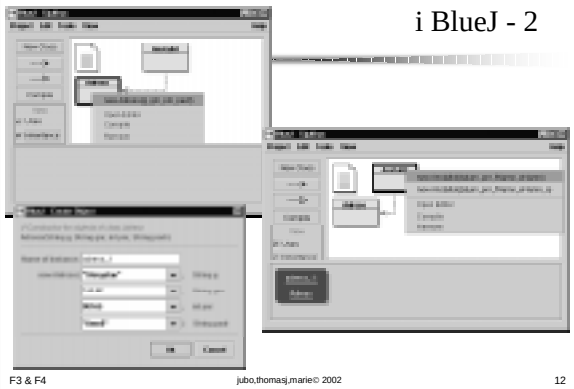
Anställd - i BlueJ



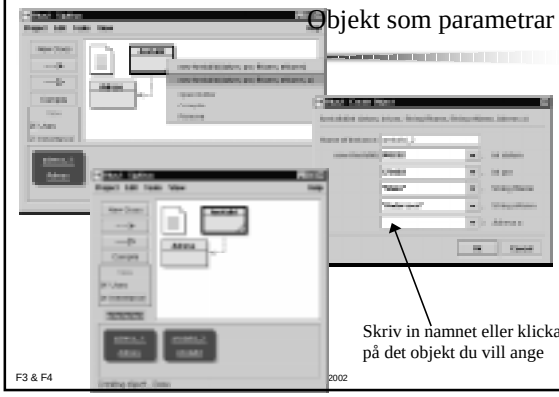
i BlueJ - 2



i BlueJ - 2



Objekt som parametrar



Innehåll

Primitiva datatyper: Hel- och flyttal

- De olika heltals och flyttals typerna har olika storlek

Typ	Storlek	Minimum	Maximum	
`byte`	8 bits	-128	127	
`short`	16 bits	-32,768	32,767	
`int`	32 bits	-2,147,483,648	2,147,483,647	
`long`	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$	
`float`	32 bits	$+/- 3.4 \times 10^{38}$	med 7 signifikanta siffror	
`double`	64 bits	$+/- 1.7 \times 10^{308}$	med 15 signifikanta siffror	

Primitiva datatyper: Tecken

- En variabel av datatypen `char` sparar precis ett s k ***Unicode tecken***
- Värdemängden är en ordnad uppräkning av tecken
- Det finns 65,536 unika Unicode tecken (16 bit) med tecken och symboler från olika språk, t ex `'å'` och `'ö'`
- Alla tecken är ordnade och varje tecken motsvarar en siffra (detta underlättar att kolla om ett tecken ligger t ex mellan `'a'` och `'z'`)
- Se Appendix B (s.708) för detaljer

Primitiva datatyper: Boolean

- En variabel av datatypen `boolean` innehåller antingen värdet `true` eller värdet `false`
- Används ofta för att definiera villkor ➜ logik
- Det finns tre logiska operatorer i Java ***inte*** (`!`), ***och*** (`&&`) och ***eller*** (`||`)

a	!a
true	false
false	true

a	b	a && b	a || b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Konstanter

- Variabler som får inte ändras
- Markeras med reserverade ordet `final`
- Guideline: Använd bara stora bokstäver
- Exempel:

```
final double PI = 3.14159;
final char NEWLINE = '\n';
```

- Fördelar
  - Vettiga namn istället för konstiga siffror eller tecken
  - Lättare att förstå koden
  - Enklare att uppdatera

Innehåll

- OOP snabbintroduktion
- Programvaruutveckling och programmering
- Datatyper
- Uttryck
- Satser
- Arv, polymorfi och dynamisk bindning
- Design (CRC)
- Metodik och klassdesign
- Att organisera Javakod
- Fält
- Undantag
- In-/utmatning och filer
- Grafik
- Rekursion
- GUI:s
- Applets vs applikationer
- Interfaces och sortering
- Noggrannheten i beräkningar

Uttryck 1

Ett ***uttryck*** är en programkonstruktion som "producerar" (evalueras till) ett resultatvärde av en viss datatyp.

Ett uttryck kan (bl a) vara:

- Ett primitivt värde (***literal***)
  - `-5, 1.23e2, 'c', true`
- En variabel
  - `smallBlueRectangle`, `PI`
- En metodanrop
  - `aTriangle.changeSize(50, 100);`
- En tilldelning
  - `height = newHeight`

Uttryck 2

Ett ***uttryck*** är en syntaktiskt korrekt kombination av literaler, variabler, metodanrop, tilldelningar och ***operatorer***.

Typiska exempel på operatorer:
Aritmetiska, t ex `+, -, *, /, …`
Logiska: `!, &&, ||`
Relationala, t ex `==, !=, <, >=, …`

Uttryck kan vara komplext
Priotitetsordning för att bestämma hur det ska evalueras

Se Appendix B (s.706) för en fullständig lista

Relations operatorer

`==`	Identiskt lika
`!=`	inte identiskt lika
`<`	mindre än
`<=`	mindre än eller lika med
`>`	större än
`>=`	större än eller lika med

ETT tecken

Prioritetsordning

- Operatorer med högre prioritet evalueras först
- Vid samma prioritet avgör associativiteten hos operatorerna (vänster-höger / höger-vänster)

Uttryck: `5 + 12 / 5 - 10 % 3`

Evalueringsordning: ③ ① ④ ②

Resultat: `6`

- Använd parenteser för att vara på säkra sidan
  - ➨ `(5 + (12 / 5)) - (10 % 3)`

Exempel

Uttryck	Resultat
`2 + 3 * 4 / 2`	`8`
`3 * 13 + 2`	`41`
`(3 * 13) + 2`	`41`
`3 * (13 + 2)`	`45`
`4 * (11 - 6) * (-8 + 10)`	`40`
`(5 * (4 - 1)) / 2`	`7`

Uttryck 3

Resultatvärdets datatyp hos ett uttryck beror på operatorn och datatyperna hos operanderna.

Uttryck	Resultat
`17 / 5`	`3`
`17.0 / 5`	`3.4`
`17 / 5.0`	`3.4`
`9 / 12`	`0`
`9 / 12.0`	`0.75`
`6 % 2`	`0`
`7 % 2.5`	`2.0`
`-7 % 2.5`	`-2.0`

```
int  dollar = 25;
float money;

money = dollar;
// OK, money blir 25.0

dollar = money;
// FEL, typerna är
// icke kompatibla
```


Innehåll

Satser

Satserna (***statements***) är grundinstruktionerna i ett programspråk.

- Block används för att gruppera satser

```
{
   sats;
   sats;
   …
}
```

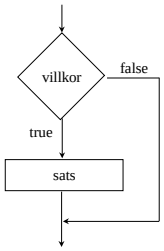
- OBS! Varje sats kan i sin tur vara ett block

Ordning på satserna

- Sekvens
  - Satserna exekveras en efter en
  - ➨ Standard
- Urval
  - Man väljer om satser ska exekveras eller ej
  - ➨ If-sats, if-else-sats, switch-sats
- Repetition
  - Satsen exekveras ett visst antal gånger
  - ➨ while-loop, do-loop, for-loop

`If`-satsen

```
if (villkor)
   sats;
```



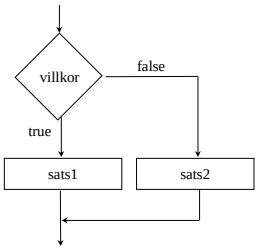
Villkor

- Villkoret är ett uttryck
- Måste evaluera till `true/false`
- ➨ Ett s k ***boolskt uttryck***

- Kan vara mer eller mindre komplicerat
- Exempel:
  - `i == j`
  - `!(a > b) && (i == j+5-k)`
  - …

If-else-satsen

```
if (villkor)
    sats1;
else
    sats2;
```



Switch-satsen

```
switch (expression)
{
   case value1:
      sats1;
      break;

   case value2:
      sats2;
      break;

   …

   default:
      satsN;
}
```

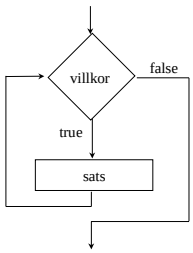
OPTIONAL

Uttrycket måste vara heltal eller tecken
Motsvarar nästade if-satser

While-satsen

```
while (villkor)
    sats;
```

- Gå runt i "varv"
  - Villkoret evalueras på nytt ***före varje varv***
  - ➨ Satsen utförs/utförs inte en gång till
- Villkoret testas *inte* kontinuerligt
  - ➨ Satsen utförs alltid helt



Oändliga loopar

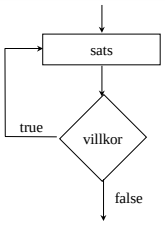
- Det får hända att satsen inte exekveras
- Villkoret måste någon gång bli falskt, annars kommer man inte ut ur loopen
- Vanligt logiskt fel
- Se t ex

```
int count = 1;

while (count <= 25)
{
   System.out.println (count);
   count = count - 1;
}

System.out.println ("Done");  // exekveras aldrig
```


Do-satsen

```
do
   sats
while (villkor);
```

- Motsvarar "omvänd" while-sats
- Villkoret evalueras på nytt ***efter varje varv***
- ➨ Satsen utförs minst en gång



For-satsen

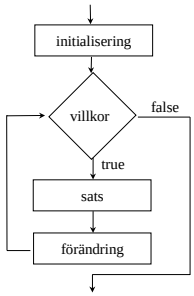
```
for (initialisering; villkor; förändring)
   sats;
```

motsvarar

```
initialisering;
while (villkor)
{
   sats;
   förändring;
}
```

Fast loop
`Initialisering`, `villkor` och `förändring` brukar refererar till samma variable
Loop-variabeln bör inte ändras i satserna

For-satsen

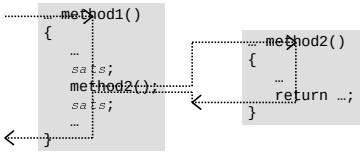


Exempel

```
for (int antal=1; antal < 75; antal++)
   System.out.println (antal);


for (int num=5; num <= total; num = num*2)
{
   sum = sum + num;
   System.out.println (sum);
}
```


Programflödet

- Programflödet är ***sekventiellt***
- Sats efter sats i den beskrivna ordningen
- Återhoppet sker till anropsstället



Referenser … en gång till ...

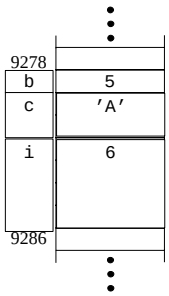
- Primitiva datatyper
  - Variablernas storlek är fastlagd
  - Alla variabler av samma datatyp tar samma utrymme i minnet
  - ➨ Värdet kan sparas "direkt"
- Klasstyper
  - Storlek av variabler varierar
  - Variabler av samma datatyp kan ta olika utrymme i minnet
  - ➨ Värdet kan inte sparas "direkt"
  - ➨ Referenser ("pekare")

➨ Olika effekt (semantik) för tilldelning (=), parameteröverföring och jämförelser (t ex ==)

Minnet 1

- Minnet består av en rad celler
- Alla celler har samma storlek (en byte)
- Varje cell har en unik adress
- En variabel är i princip ett (vettigt) namn på en adress
- Stora värden sparas i konsekutiva celler
- Exempel (primitiva datatyper):

```
byte b = 5;
char c = 'A';
int i = 6;
```



Minnet 2

- Referenser är också adresser
- Exempel (klasstyper):

```
Triangle t1 =
   new Triangle();

Triangle t2 =
   new Triangle();
```