Innehåll

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

Att tänka rekursivt

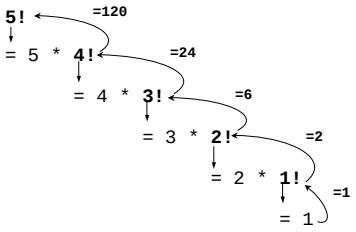
- Rekursion är en grundläggande programmeringsteknik
- M h a rekursion kan vissa problem lösas på ett mycket elegant sätt
- I en rekursiv definition används ordet eller konceptet som definieras i själva definitionen
- Ganska vanlig inom t ex matematik
  - Fakultetsfunktion

$$n! = \begin{cases} 1 & \text{om } n = 0 \\ n \times (n-1)! & \text{om } n > 0 \end{cases}$$

  - Fibonacci tal

$$FIB(n) = \begin{cases} 1 & \text{om } n <= 2 \\ FIB(n-1) + FIB(n-2) & \text{om } n > 2 \end{cases}$$

Ett exempel



En rekursiv listdefinition

- En lista är ett listelement följd av en lista, t ex

$$\texttt{Nummerlista} = \begin{cases} \texttt{nummer} \\ \texttt{nummer } \textit{följd av } \texttt{Nummerlista} \end{cases}$$

- Inklusive tomma listor

$$\texttt{Nummerlista} = \begin{cases} \texttt{[]} \\ \texttt{nummer } \textit{följd av } \texttt{Nummerlista} \end{cases}$$

➨ `Nummerlista` definieras m h a sig själv

Ett exempel

- Listelement åtskiljs med ett komma

```
Lista = { nummer
        { nummer komma Lista
```

Den icke-rekursiva delen (basfallet) ***terminerar*** (avslutar) definitionen
T ex definieras listan `24, 88, 40, 37` genom

```
nummer komma Lista
  24      , 88, 40, 37
          nummer komma Lista
            88      , 40, 37
                    nummer komma Lista
                      40      , 37
                              nummer
                                37
```


Rekursion allmänt

```
LösRekursivt (problem):

   Om problemet är trivialt:
      Lös problemet m h a basfallet;

   Annars:   // när problemet är icke-trivialt
      Förenkla problemet;  // dela upp

      För alla delproblem do:
         LösRekursivt (delproblem);

      Sätt ihop lösningarna till delproblemen
      till en lösning för originalproblemet;
```


Fibonacci exemplet

Definition

```
FIB(n) = { 1                   om n <= 2
         { FIB(n-1) + FIB(n-2) om n > 2
```

Som Java metod

```
public int fibonacci (int n)
{
   if (n <= 2)   // basfall
      return 1;
   else
      return (fibonacci(n-1) + fibonacci(n-2));
}
```


Rekursiva metoder

- En metod i Java får anropa sig själv
- En rekursiv metod måste hantera
  - Basfallet
  - Direkta eller indirekta rekursiva anrop
- Varje anrop exekveras i ett eget miljö med nya parametrar och nya lokala variabler
- ➨ ***Inkarnation*** av metoden
- När en inkarnation avslutas återges kontrollen till anroparen (som kan vara en inkarnation av samma metod)

Ett exempel

- Beräkna summan av alla tal mellan ett och N∈N
- Problemet kan definieras rekursivt

$$\text{Summan} = \sum_{i=1}^{N} = N + \sum_{i=1}^{N-1} = N + (N-1) + \sum_{i=1}^{N-2} = \ldots$$

```
public int summa (int N)
{
   int result;     // hjälpvariabel
   if (N == 1)     // basfall
      result = 1;
   else
      result = N + summa(N-1);
   return result;
}
```


Rekursion vs iteration

- Alla problem går att lösa iterativt, t ex `summa`

```
public int summa (int N)
{
   int result = 0;   // hjälpvariabel
   for (int i=1; i <= N; i++)
      result = result + i;
   return result;
}
```

- Rekursiva lösningar är lämpliga om
  - Datat som hanteras har definierats rekursivt
  - Problemet är rekursivt
- Rekursiva lösningar är oftast elegantare/ enklare
- Iterativa lösningar är oftast effektivare

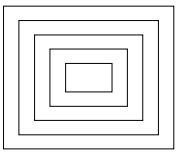
Palindrom

- Ett ***palindrom*** är en sträng som är lika baklänges
  - radar
  - able was I ere I saw elba
- Ett palindrom test kan enkelt definieras rekursivt
- ➨ OK, om yttersta tecknen är lika och resten är palindrom

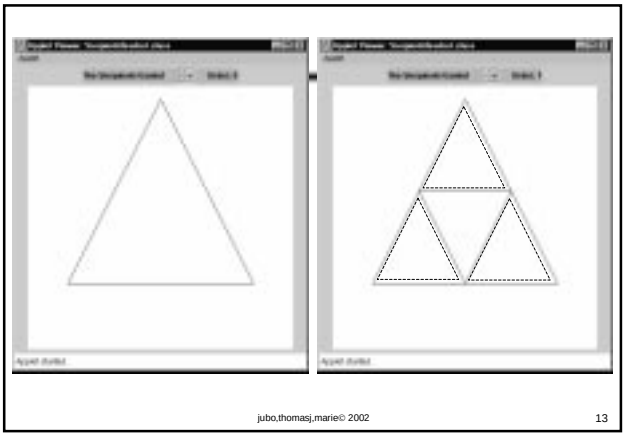
```
public boolean palindrom (String str)
{
   if (str.length() <= 1)
      return true;
   else
   {
      if (str.charAt(0) == str.charAt(str.length()-1))
         return (palindrom
            (str.substring (1, str.length()-2));
      else
         return false;
   }
}
```

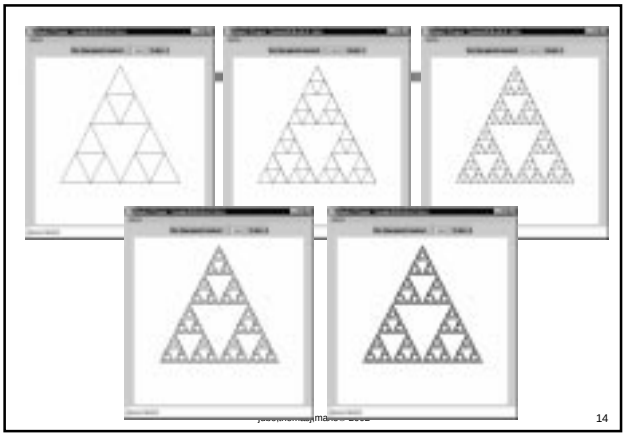

Rekursiva bilder

- Rekursion lämpar sig bra om problemet går lätt att beskriva rekursivt, t ex
  - Själv repeterande mönster

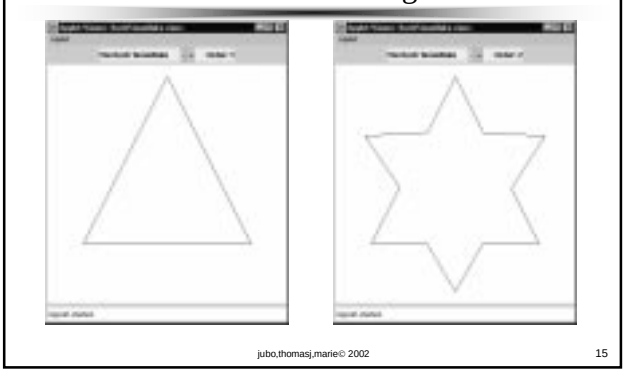


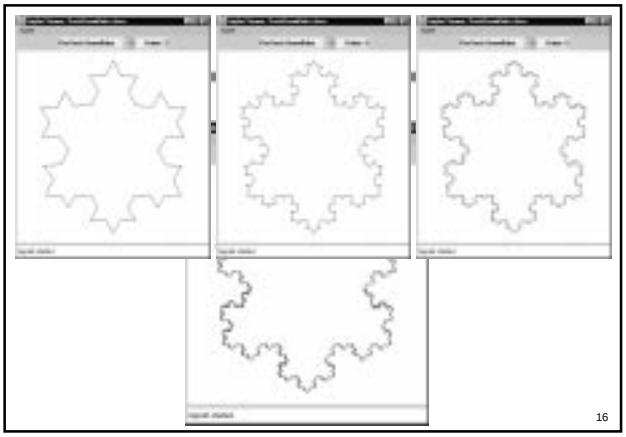
- ➨ Ganska komplicerade att göra iterativt





Koch’s snöflinga



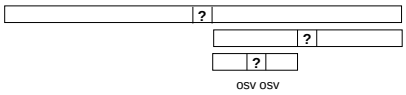


Sökning

- I ***osorterade fält*** används linjär sökning:
  - Går igenom fältet element för element
  - Är det aktuella elementet det sökta ➜ OK
  - Annars går till nästa element
- Om fältet är sorterade kan sökningen avbrytas när det aktuella elementet blir för stort/ litet

Binärsökning

- Om fältet är ***sorterat*** kan binärsökning användas
- Idé: Dela och härska
  - Kolla upp mellersta elementet
  - Tre fall
    - Sökelement hittat ➜ OK
    - Sökelement större ➜ Fortsätt på samma sätt med högra delen
    - Sökelement mindre ➜ Fortsätt på samma sätt med vänstra delen
  - Fortsätt tills hittat eller inga fler delningar möjliga



osv osv

- Lämpar sig bra för rekursiv lösning