Java Threads

- Möjliggör mer än en process eller "task" samtidigt
- Ett sekventiellt flöde genom ett program samtidigt som ett annat sekventiellt flöde
- En tråd kan exekvera satserna i ett program samtidigt som en annan tråd exekverar satserna i samma program

Skapa trådar

- Göra `extend` på klassen `Thread` och ersätta metoden `run`
- Alternativt skapa en klass som implementerar interfacet `Runnable`

Exekvera trådar

- Det är i metoden `run` som man talar om vad som ska göras, men man startar en tråd genom anrop av `start`

Synkronisering

- Modifieraren `synchronized` används
- Förhindrar att delade data används samtidigt av olika trådar
- När en synkroniserad metod används låses objektet så att inga andra trådar kan exekvera några metoder för objektet

Programspråksteori, sammanfattning och slutsatser

- Varför programspråksteori?
- Viktiga kriterier
- Språkjämförelser
- Språkdesign

Nyttan med programspråksteori

- Kunskap om arbetsredskapet
- Bredda tänkandet kring design och konstruktion
- Kriterier för språkval

Strategiska aspekter

- Abstraktion
- "Programming in the large"
- Återanvändning

Taktiska aspekter

- Modellering
- Nivå
- Säkerhet
- Effektivitet
- Kompilator
- Förtrogenhet

Språkjämförelser

- Störst skillnad mellan olika paradigmer
- Jämför olika typer av problem

Java: Quicksort

```
public static void quickSort( int[]
  number, int low, int high )
{
   if ( low < high ) {

      int mid = partition( number, low,
       high );

      quickSort( number, low, mid-1 );
      quickSort( number, mid+1, high );
   }
}
```


Java: Partition

```
private static int partition( int[] number, int start,
   int end )
{
    int pivot = number[start];
    do {
        while ( start < end && number[end] >= pivot)
        end--;
        if ( start < end ) {
            number[start] = number[end];
            while ( start < end && number[start] <=
              pivot)
                start++;
            if (start < end)
                number[end] = number[start];
        }
    } while ( start < end );
    number[start] = pivot;
    return start;
}
```


ML: Quicksort

```
fun sort (nil) =
      nil
   | sort [singleitem] =
      [singleitem]
   | sort (firstitem::otheritems) =
      let val (smallitems, bigitems) =
            partition (firstitem,
             otheritems)
      in
         sort (smallitems) @ [firstitem]
          @ sort (bigitems)
      end
```

ML: Partition

```
fun partition (pivot, nil) =
      (nil, nil)
  | partition (pivot, first::others) =
      let val (smalls, bigs) =
            partition (pivot, others)
      in
        if first < pivot
        then (first::smalls, bigs)
        else (smalls, first::bigs)
      end
```

Språkdesign

- val av begrepp
- regelbundenhet
- enkelhet
- syntax

Begrepp

I alla programspråk idag bör följande förekomma:

- Värden
- Bindningar
- Abstraktion

- Generella språk behöver kommandon och variabler
- Även inkapsling, ”packages”

Regelbundenhet

- Typkompletthetsprincipen (type completeness principle)
- Abstraktionsprincipen (abstraction principle)
- Principen för överensstämmelse (correspondence principle)
- Kvalifikationsprincipen (qualification principle)

Enkelhet

- Hjälp att lösa problem
- Naturliga lösningar
- Begränsad mängd kraftfulla begrepp
- Liten kärna med kraftfulla abstraktionsmekanismer

Syntax

- Av sekundär betydelse, men inte oviktigt
- God syntax kännetecknas av:
  - Läsbarhet
  - Överensstämmelse mellan semantiska begrepp och syntax
  - Lätt att lära

Kort sagt:

- Enkelhet
- Säkerhet
- Effektivitet
- Läsbarhet