TDA 545: Objektorienterad programmering

Föreläsning 4:
for, while, do-while

Magnus Myréen

Chalmers, läsperiod 1, 2015-2016

Frågor efter förra föreläsning

Vad betyder:

```
x = x+1
x++
++x
x == x+1
x != x+1
```

special uttryck med ***sidoeffekt***: uppdatering av variabeln x

en tilldelningssats (assignment) där x uppdateras till värdet av uttrycket x+1

ett uttryck, värdet är x, men variabeln x uppdateras också till värdet av uttrycket x+1

ett uttryck, först uppdateras x till x+1, sedan returneras värdet av x, dvs nu x+1

ett uttryck av typen boolean, testar (o)likhet mellan två heltals uttryck (int expressions)

Fråga mera! Det här är mycket bra frågor. *Uppskattas!*

Kan man skriva test som visar vad svaren är?

while loopar

Blocket utförs **noll** eller flera gånger.

Syntax

while (<boolean expression>) {
    <statements>
}

Informell semantik

1: beräkna <boolean expression>
2: om det blev true, kör <statements>
   och fortsätt från steg 1 igen.
3: om det blev false, då är while klart,
   dvs programmet fortsätter köra koden
   som kommer efter }

Tips! **I beräkningar** kan man ersätta

```
while (<bool-exp>) { satser }
```

med

```
if (<bool-exp>) {
  satser;
  while (<bool-exp>) { satser }
}
```

Exempel för while

```
int i = 0;
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Exempel för while

```
int i = 0;
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Exempel för while

```
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

typ: `int`
namn: `i`

Exempel för while

```
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

typ: int
namn: i

Exempel för while

```
if (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

typ: `int`
namn: `i`

Exempel för while

```
if (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

typ: `int`
namn: `i`

Exempel för while

```
if (0 < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

typ: `int`
namn: `i`

Exempel för while

```
if (true) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

typ: `int`
namn: `i`

Exempel för while

```
System.out.print(i + " ");
i = i + 1;
while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
}
```

typ: `int`
namn: `i`

Exempel för while

```
System.out.print(i + " ");
i = i + 1;
while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
}
```

typ: `int`
namn: `i`

Exempel för while

```
System.out.print(i + " ");
i = i + 1;
while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
}
```

typ: `int`
namn: `i`

Exempel för while

```
System.out.print(0 + " ");
i = i + 1;
while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
}
```

typ: `int`
namn: `i`

Exempel för while

```
System.out.print("0 ");
i = i + 1;
while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
}
```

typ: `int`
namn: `i`

Exempel för while

```
System.out.print("0 ");
i = i + 1;
while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
}
```

typ: `int`
namn: `i`

Exempel för while

```
i = i + 1;
while (i < 3) {
   System.out.print(i + " ");
   i = i + 1;
}
```

typ: `int`
namn: `i`

Utskrift (output):

“0 ”

Exempel för while

```
i = i + 1;
while (i < 3) {
   System.out.print(i + " ");
   i = i + 1;
}
```

typ: `int`
namn: `i`

Utskrift (output):

“0 ”

Exempel för while

```
i = 0 + 1;
while (i < 3) {
   System.out.print(i + " ");
   i = i + 1;
}
```

Utskrift (output):

“0 ”

Exempel för while

```
i = 1;
while (i < 3) {
   System.out.print(i + " ");
   i = i + 1;
}
```

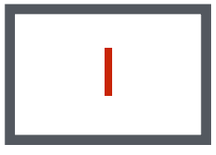
typ: `int`
namn: `i`

Utskrift (output):

“0 ”

Exempel för while

typ: `int`
namn: `i`




```
while (i < 3) {
   System.out.print(i + " ");
   i = i + 1;
}
```

Utskrift (output):

“0 ”

Exempel för while

typ: `int`
namn: `i`

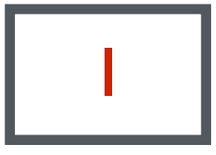

```
while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
}
```

Utskrift (output):

“0 ”

Exempel för while

typ: `int`
namn: `i`



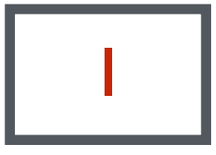

```
if (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Utskrift (output):

“0 ”

Exempel för while

typ: `int`
namn: `i`




```
if (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Utskrift (output):

“0 ”

Exempel för while

typ: `int`
namn: `i`

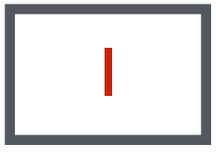

```
if (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Utskrift (output):

“0 ”

Exempel för while

typ: `int`
namn: `i`



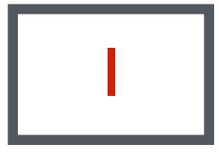

```
if (1 < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Utskrift (output):

“0 ”

Exempel för while

typ: `int`
namn: `i`




```
if (true) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Utskrift (output):

“0 ”

Exempel för while

typ: `int`
namn: `i`


```
System.out.print(i + " ");
i = i + 1;
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Utskrift (output):

“0 ”

Exempel för while

typ: `int`
namn: `i`


```
System.out.print(i + " ");
i = i + 1;
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Utskrift (output):

“0 ”

Exempel för while

typ: `int`
namn: `i`


```
i = i + 1;
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Utskrift (output):

“0 1 ”

Exempel för while

typ: `int`
namn: `i`


```
i = i + 1;
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Utskrift (output):

“0 1 ”

Exempel för while

typ: `int`
namn: `i`


```
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Utskrift (output):

“0 1 ”

Exempel för while

typ: `int`
namn: `i`


```
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Utskrift (output):

“0 1 ”

Exempel för while

typ: `int`
namn: `i`


```
if (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Utskrift (output):

“0 1 ”

Exempel för while

typ: `int`
namn: `i`


```
if (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Utskrift (output):

“0 1 ”

Exempel för while

typ: `int`
namn: `i`


```
if (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Utskrift (output):

“0 1 ”

Exempel för while

typ: `int`
namn: `i`


```
if (true) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Utskrift (output):

“0 1 ”

Exempel för while

typ: `int`
namn: `i`


```
System.out.print(i + " ");
i = i + 1;
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Utskrift (output):

“0 1 ”

Exempel för while

typ: `int`
namn: `i`


```
System.out.print(i + " ");
i = i + 1;
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Utskrift (output):

“0 1 ”

Exempel för while

typ: `int`
namn: `i`


```
i = i + 1;
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Utskrift (output):

“0 1 2 ”

Exempel för while

typ: `int`
namn: `i`


```
i = i + 1;
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Utskrift (output):

“0 1 2 ”

Exempel för while

typ: `int`
namn: `i`


```
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Utskrift (output):

“0 1 2 ”

Exempel för while

typ: `int`
namn: `i`


```
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

Utskrift (output):

“0 1 2 ”

Exempel för while

typ: `int`
namn: `i`

Utskrift (output):

“0 1 2 ”

```
if (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Exempel för while

typ: `int`
namn: `i`

Utskrift (output):

“0 1 2 ”

```
if (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Exempel för while

typ: `int`
namn: `i`

Utskrift (output):

“0 1 2 ”

```
if (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Exempel för while

typ: `int`
namn: `i`

Utskrift (output):

“0 1 2 ”

```
if (3 < 3) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Exempel för while

typ: `int`
namn: `i`

detta betyder att koden mellan { … } inte körs

```
if (false) {
  System.out.print(i + " ");
  i = i + 1;
  while (i < 3) {
    System.out.print(i + " ");
    i = i + 1;
  }
}
```

Utskrift (output):

“0 1 2 ”

Exempel för while

typ: `int`
namn: `i`

Utskrift (output):

“0 1 2 ”

koden har kört klart

Exempel 2

```
int i = 0;
while (i != 3) {
   System.out.print(i + " “);
   i = i + 2;
}
```

vad händer om vi ändrar koden så här?

och så här?

Exempel 2

```
int i = 0;
while (i != 3) {
  System.out.print(i + " “);
  i = i + 2;
}
```

Utskrift (output):

```
0 2 4 6 8 10 … 2000000 2000002 2000004 … osv
```

Att skriva egan loopar

Att se hur loopar kör är *relativt lätt*.

Att skriva egna loopar kräver mycket övning!

Utmaning:

Skriv ett program som räknar fakulteten av ett tal.

Matematik: $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$

En lösning

```
public class Fac {
    public static void main(String[] args) {

        int k = 5;
        int result = 1;
        int i = 1;
        while (i <= k) {
            result = result * i;
            System.out.println("fac(" + i + ") = " + result);
            i = i+1;
        }

    }
}
```

En utmaning till

Utmaning:

Räkna *greatest common divisor* (*GCD*) av två heltal.

Matematik: gcd(45,35) = 5

En lösning

```
public class GCD {
    public static void main(String[] args) {

        int k = 45;
        int l = 35;
        while (k != l) {
            if (l < k) { k = k - l; } else { l = l - k; }
            System.out.println("k = " + k);
            System.out.println("l = " + l);
        }
        System.out.println("Result: " + k);

    }
}
```

Exempel 2

Loopar av denna stil är mycket vanliga:

```
int i = 0;
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}
```

deklaration och utgångsvärde

villkor för while

beräkning som inte ändrar på **i**

justering av variabeln **i**

Dessa loopar kan skrivas med **for**-satsen:

deklaration och utgångsvärde

villkor för while

justering av variabeln **i**

```
for (int i = 0; i < 3; i = i + 1) {
  System.out.print(i + " ");
}
```

beräkning som inte ändrar på **i**

```
int i = 0;
while (i < 3) {
  System.out.print(i + " ");
  i = i + 1;
}

for (int i = 0; i < 3; i = i + 1) {
  System.out.print(i + " ");
}
```

Tips! For av formen

```
for (<init>, <bool>, <chg>) {
  <stmts>
}
```

är precis samma som

```
<init>;
while (<bool>) {
  <stmts>;
  <chg>
}
```

for satsen: loopar med känt antal varv

Syntax

```
for(<init>;  <boolExpr>;  <change>) {
   <statements>
}
```

Tips! För av formen

```
for (<init>, <bool>, <chg>) {
  <stmts>
}
```

är precis samma som

```
<init>;
while (<bool>) {
  <stmts>;
  <chg>
}
```

Informell Semantik

1. "`init`" uttrycket beräknas (sker bara en gång). Vanligen initialiserar man bara en räknare till dess startvärde.
2. `boolExpr` beräknas.
   Så länge det är sant kommer
3. loopens satser (`statements`) att beräknas.
4. uttrycket `change` beräknas. Vanligen ökar man räknaren med ett. Gå till punkt 2.

- räknaren skall deklareras. Om det görs i loopen så är den inte känd utanför loopen
- räknaren bör inte ändras i loopens satser
- `init`, `boolExpr`, `change` kan uteslutas
- `init` och `change` kan vara flera komma-separerade satser

Inget av de två sista rekommenderas

Exempel på for-satsen

```
for (int i = 0; i <= 8; i = i+1) {
  System.out.print(i + " ");
}
// 0 1 2 3 4 5 6 7 8  (på en rad)
```

använd endast den här formen

```
for (char ch = 'a'; ch <= 'z'; ch++) {
  System.out.println(ch);
}
// a b c .... z med en per rad
```

```
for(int tal=1; tal<=50; tal=2*tal){
  System.out.println(tal);
}
// tal blir 1, 2, 4, 8, 16, 32
```

Keep it simple!

Att skriva egan loopar

Utmaning:

Implementera fakultet exemplet med en **for** loop.

Matematik: $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$

Tentamensuppgift

Uppgift 3: [5 poäng] *loopar, utskrift*

Skriv ett program som skriver ut multiplikationstabellerna upp till tio. Exempel utskrift:

```
1  2  3  4  5  6  7  8  9  10
2  4  6  8  10 12 14 16 18 20
3  6  9  12 15 18 21 24 27 30
4  8  12 16 20 24 28 32 36 40
5  10 15 20 25 30 35 40 45 50
6  12 18 24 30 36 42 48 54 60
7  14 21 28 35 42 49 56 63 70
8  16 24 32 40 48 56 64 72 80
9  18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

Extra: hur kan man få utskriften att se snyggare ut?

En lösning

```
public class Mult {
    public static void main(String[] args) {

        for (int i=1; i<=10; i++) {
            for (int j=1; j<=10; j++) {
                System.out.print((i*j) + " ");
            }
            System.out.println();
        }

    }
}
```

do-while loopar

Blocket utförs **en** eller flera gånger

***** Ovanligt att man vill göra det! *****

Syntax

do {
    <statements>
} while (<boolean expression>);

Informell Semantik

1. Satserna (`statements`) utförs en gång
2. Loopvillkoret (`boolean expr.`) beräknas.
   Om det är sant utförs 1-2 igen
3. Om villkoret är falskt görs ingenting dvs loopen terminerar.

Tips! do-while av formen

```
do {
  <stmts>
} while (<bool-exp>);
```

är precis samma som

```
<stmts>;
while (<bool-exp>) {
  <stmts>
}
```

Scanner (används i nästa ex.)

Klassen java.util.Scanner förser oss med inmatningsprimitiver
så vi kan läsa “tokens" från tangentbordet:

(en token kan vara ett tal eller en sträng)

```
import java.util.Scanner;
...
Scanner in = new Scanner(System.in);
String str;
while (in.hasNext()) {
  str = in.next();
  System.out.println(str);
}
```

Metoder i `Scanner`: (ersätt `X` med t.ex. `Int`)

- `in.hasNextX()` finns det ett `X`
- `in.hasNext()` finns det någonting
- `in.nextX()` ger nästa `X`
- `in.next()` ger nästa token som sträng
- `in.nextLine()` ger hela raden som en sträng

do-while exempel

```
int n = 0;
do {
  println(“Mata in ett tal >5 ”);
  n = in.nextInt();
} while (n <= 5);
```

precis samma som:

```
int n = 0;
println(“Mata in ett tal >5 ”);
n = in.nextInt();
while (n <= 5) {
  println(“Mata in ett tal >5 ”);
  n = in.nextInt();
}
```

Vanliga fel vid konstruktion av loopar

De vanligaste felen är oändliga loopar och off-by-one fel.

(pseudo-)**oändlig loop**:

```
int i = 0;
while (i < 10) {
  satser ...
  i = i-1;
}
```

För att komma ur en oändlig loop tryck: control-C (dvs håll in tangenten ctrl och tryck på C)

off-by-one: (arrays kommer senare)

```
int[] array = new int[9];
for (int i=0; i<=9; i=i+1) {
  array[i] = läs tal från användaren
}
```

försöker läsa index 9 som inte finns!

Kontrollstrukturer

När använder man vad?

Några generella tips:

for:

Loopar med **känt** antal varv

”keep it simple” - det bör vara nåt i stil med

```
for (int i=0, i<10; i++) ....
```

använd annars en while

while

Loopar med **okänt** antal varv

noll eller flera varv

do-while

Loopar med okänt antal varv

ett eller flera varv (ovanligt! - undvik)

De flesta satserna är avklarade

Ett program byggs huvudsakligen upp av
deklarationer och **satser**

Enkla satser

(enkla satser avslutas med ";")

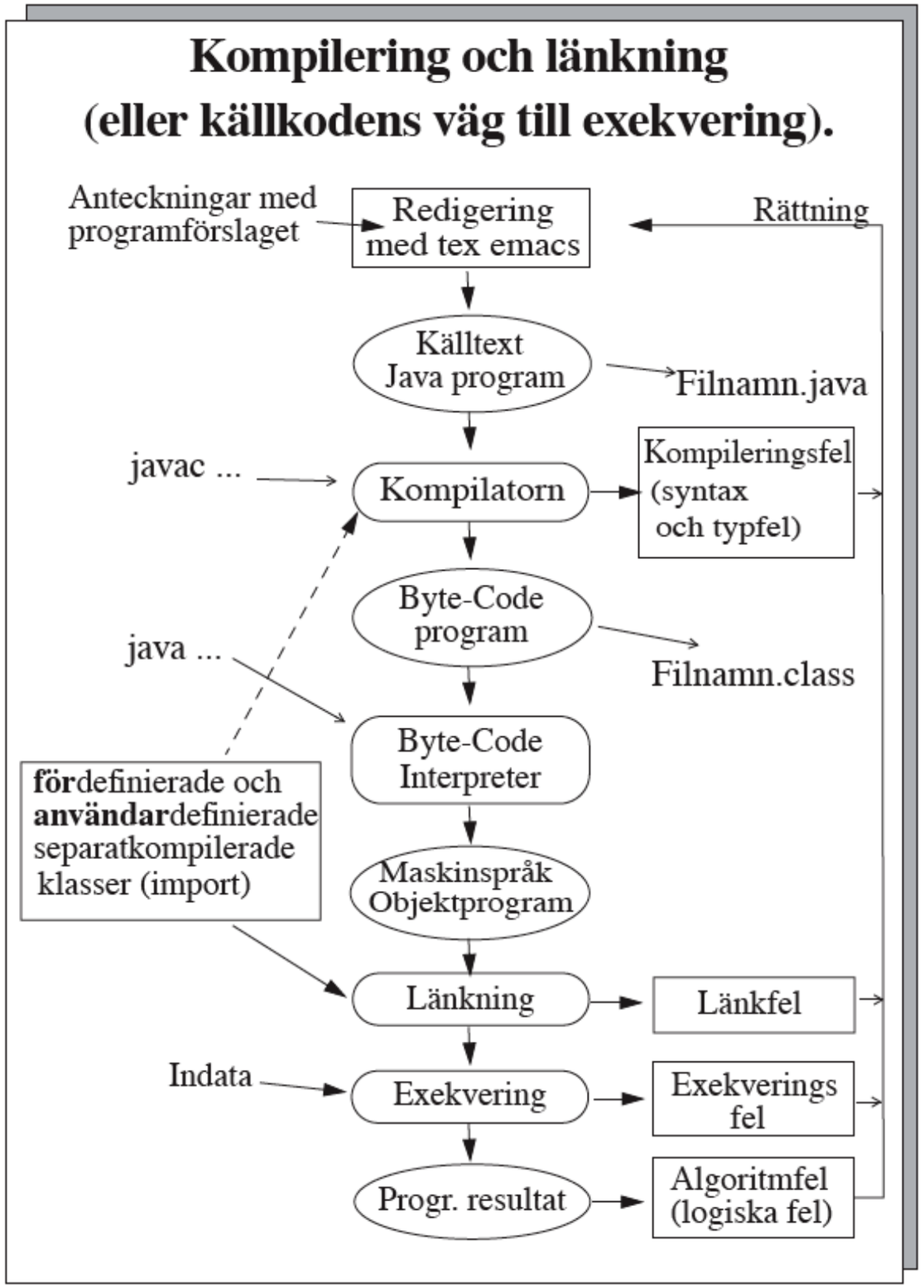
- Tilldelning
- Metodanrop (av void metoder)
- break ("Kortslutning" av loop/switch sats)
- return <uttryck>
  (För att avsluta underprogram eller ge resultat från funktioner)
- ; (Gör ingenting !)

Sammansatta satser

(avslutas INTE med ";" utom do-while)

- Villkorsats (if)
- Flervalsats (switch)
- Iterationssatser (for, while och do-while)
- Blocksats ({satser})

Kompilering och länkning



Kompilering och länkning

1. Från anteckningarna skapas en sk källtext med hjälp av ett editeringsprogram tex emacs. Källtexten utgör den för oss läsbara beskrivningen av det vi vill att programmet skall göra. Program, programtext och källprogram är andra benämningar på källtext. Vanligen slutar källtextens namn på något som avslöjar vilket språk som använts, tex .ada, .java, .c (C), .p (Pascal), .asm (assembler) osv.

2. Källtexten kan vi ge som indata till en kompilator. En kompilator översätter, den för oss begripliga men för datorn obegripliga, texten till ”maskinkod” som datorn ”förstår”. Detta kallas alltså kompilering. I Javas fall så översätts källtexten till en mellanform, ett slags maskinspråk för alla datorer, som kallas bytekod. Kompilatorn analyserar också texten och kontrollerar om programmet vi skrivit är ett språkligt korrekt program. Om det tex förekommer syntaxfel eller typfel i källtexten så meddelar kompilatorn detta och stannar.

3. Som resultat av kompileringen får vi ett program som innehåller de genererade bytekodinstruktionerna. Detta lagras i en fil som heter samma som källtexten men med appendix “.class”.

4. Varje dator har sedan en “bytekod interpreter” som som översätter till den aktuella datorns maskinkod. Detta program är på maskinkodsform men alla referenser (adresser) är inte upplösta (kända). Det innebär att vissa andra programmoduler (som tex java.lang) inte är kända ännu eftersom de kompilerats separat.

5. Nästa steg är att kalla på länkaren som bestämmer var instruktionerna skall ligga i minnet och fyller i alla referenser mellan modulerna. Om vi länkar ihop våra objektprogram och andra referenser så får vi ett exekverbart program, ett laddprogram.

6. Ett laddprogram är färdigt att ”ladda” in i datorn för att köras. I detta program är alla referenser upplösta.

7. När vi exekverar programmet får vi eventuellt förse det med indata och får (eventuellt) ett resultat (utdata). Vi kan exekvera programmet genom att ge namnet på källtextfilen utan appendix.

Dessa faser måste dessutom gås igenom varje gång man ändrar något i källtexten

Kompilering, länkning och körning

Steg 1: Använd en editor för att skapa en källtextfil (source file):
I ett terminalfönster:

```
% emacs Pnrkontroll.java
```

som tillsvidare måste innehålla en `main` metod.

Steg 2: Kör Java kompilatorn (javac):

```
% javac Pnrkontroll.java
```

Skapar en fil som heter `Pnrkontroll.class`
Rätta alla fel... genom att göra om stegen ovan

Steg 3: Kör programmet (med java):

```
% java Pnrkontroll
```

OBS, ***utan*** .java eller .class på slutet

Ifall programmet fastnar i en oändlig loop: håll ner "ctrl" och C-knappen.

Indentering och att placera '{ }'

Rätt indenterat:

```
static boolean odd(int tal) {
  if (tal%2 == 0) {
    return false;
  } else {
    return true;
  }
} // end odd
```

Fel indenterat:

```
static boolean odd(int tal)
{
  if (tal%2 == 0)
  {
    return false;
  }
  else
  {
    return true;
  }
} // end odd
```

Undvik onödigt tomrum inuti metoder.

Indentering = läsbarhet + korrekthetskoll

Indentering och att placera '{ }'

Rätt indenterat:

```
static boolean odd(int tal) {
  if (tal%2 == 0) {
    return false;
  } else {
    return true;
  }
} // end odd
```

Fel indenterat:

```
static boolean odd(int tal)
{
  if (tal%2 == 0)
  {
    return false;
  }
  else
  {
    return true;
  }
} // end odd
```

Rätt och förenklat:

```
static boolean odd(int tal) {
  return (tal%2 != 0)
} // end odd
```

Fel indenterat! (oläsbart)

```
static boolean odd(int tal) { if (
tal%2 == 0) { return false; } else
{ return true; } }
```

Aritmetik med heltal (dvs int)

```
int x = 5;
int y = 3;
x = x / 3;
```

Anta att minnet är:

variabeler, dvs
minnesplatser

Aritmetik med heltal (dvs int)

```
int x = 5;
int y = 3;
x = x / 3;
```

Anta att minnet är:

variabeler, dvs
minnesplatser

Aritmetik med heltal (dvs int)

```
int y = 3;
x = x / 3;
```

Anta att minnet är:

variabeler, dvs
minnesplatser

typ: `int`
namn: `x`

Aritmetik med heltal (dvs int)

```
int y = 3;
x = x / 3;
```

Anta att minnet är:

variabeler, dvs
minnesplatser

typ: `int`
namn: `x`

Aritmetik med heltal (dvs int)

```
x = x / 3;
```

Anta att minnet är:

variabeler, dvs
minnesplatser

typ: `int`
namn: `x`

typ: `int`
namn: `y`

Aritmetik med heltal (dvs int)

```
x = x / 3;
```

Anta att minnet är:

variabeler, dvs
minnesplatser

typ: `int`
namn: `x`

typ: `int`
namn: `y`

Aritmetik med heltal (dvs int)

```
x = 5 / 3;
```

heltals division, dvs resultatet måste vara av typen **int**

Anta att minnet är:

variabeler, dvs minnesplatser

typ: `int`
namn: `x`

typ: `int`
namn: `y`

Aritmetik med heltal (dvs int)

```
x =    1;
```

heltals division, dvs resultatet måste vara av typen **int**

Anta att minnet är:

variabeler, dvs minnesplatser

typ: `int`
namn: `x`

typ: `int`
namn: `y`

Aritmetik med heltal (dvs int)

koden har kört klart

Anta att minnet är:

variabeler, dvs
minnesplatser

typ: `int`
namn: `x`

typ: `int`
namn: `y`

Programmeringsövning

Övning:

Skriv kod som implementerar heltals division.

Exempel: 7 / 2 = 3

En lösning

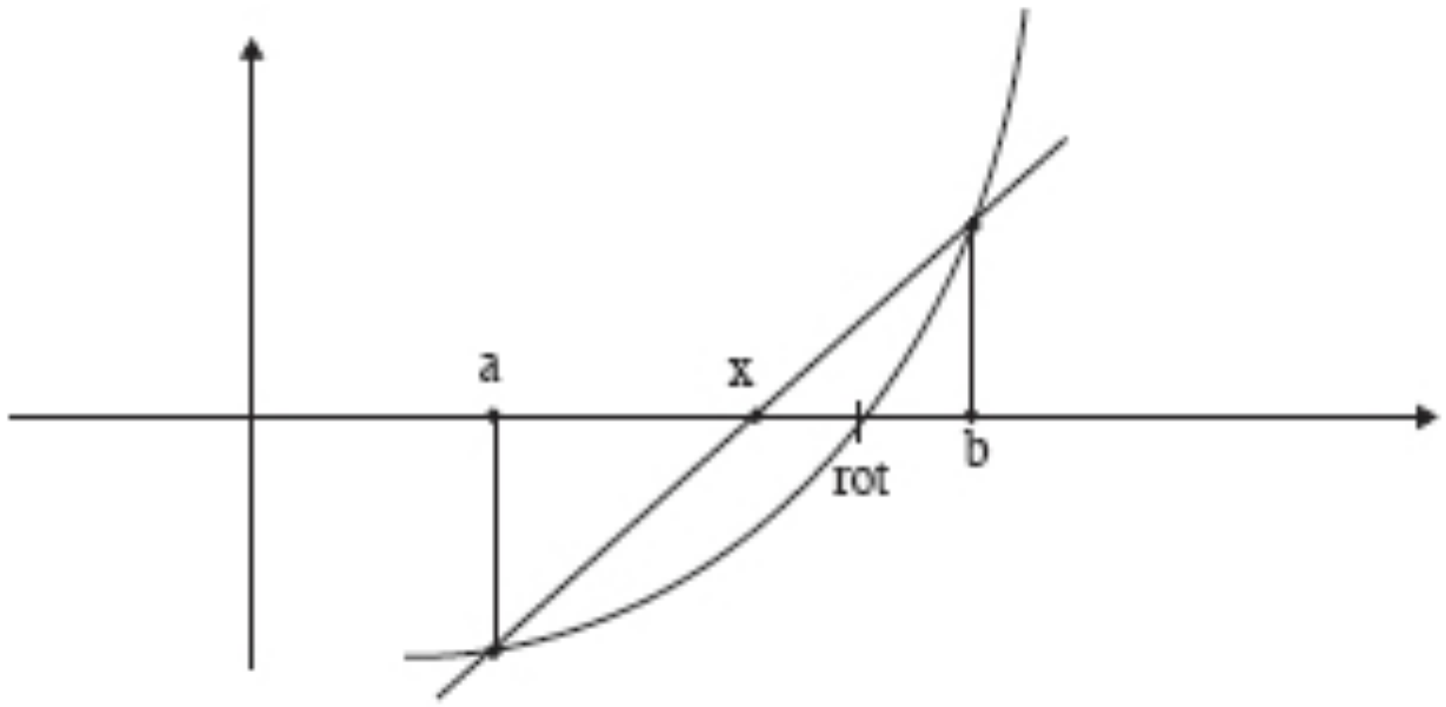
```
public class IntDiv {
    public static void main(String[] args) {

        int m = 7;
        int n = 2;

        int res = 0;
        System.out.print(m + " / " + n + " = ");
        while (n <= m) {
            m = m - n;
            res = res + 1;
        }
        System.out.println(n + " * " + res + " + " + m);

    }
}
```

Programmeringsövning 2

Sekanten:

Skriv kod som löser en enkel version av sekant labben från förra året (kommer inte i år.)

http://www.cse.chalmers.se/edu/year/2013/course/tda545/courseMtrl/assignments/lab3/secant.html



Början på en lösning

```
public class Secant {

    public static double f(double x) {
        return 0.5 * x * x - 5.0 * x + 2.0;
    }

    public static void main(String[] args) {
        double a = -5.0;
        double b = 5.0;
        double eps = 0.0001;
        double x;

        // Skriv kod här som hittar nollstället på funktionen f.
        // Lösningen kan vara en while eller do-while loop.

        System.out.println("Result: " + x);
    }

}
```

Läsanvisning

Jag pekar nedan på var i boken de olika sakerna tas upp. Boken tar dock upp dessa saker ganska sent i sammanhang med andra saker som ni inte ännu behärskar. Det kan därför vara rätt svårt att läsa dessa hänvisningar nu.

- datatypen Boolean 4.2, 4.6
- relationsuttryck och logiska uttryck (.)
- De Morgan's lag (s 217)
- operatorer (utspritt)
- if (4.1), switch(4.8)
- for (12.7,12.6),
- while (7.3.1, 12.6), do-while (7.5)
- kompilering och länkning

Nästa gång: att använda färdiga klasser kap 2.

Glöm inte att programmera, programmera, programmera!