INF1000 : Forelesning 3

Programeksempler
Løkker
Arrayer

Ole Christian Lingjærde
Biomedisinsk forskningsgruppe
Institutt for informatikk
Universitetet i Oslo

Body Mass Index (BMI)

Vi skal lage et program som beregner BMI ut fra høyde og vekt og gir melding om hvilken vektstatus det tilsvarer:

BMI	Vektstatus
Under 18.5	Undervekt
18.5 – 24.9	Normal vekt
25.0 – 29.9	Overvekt
30.0 eller høyere	Fedme

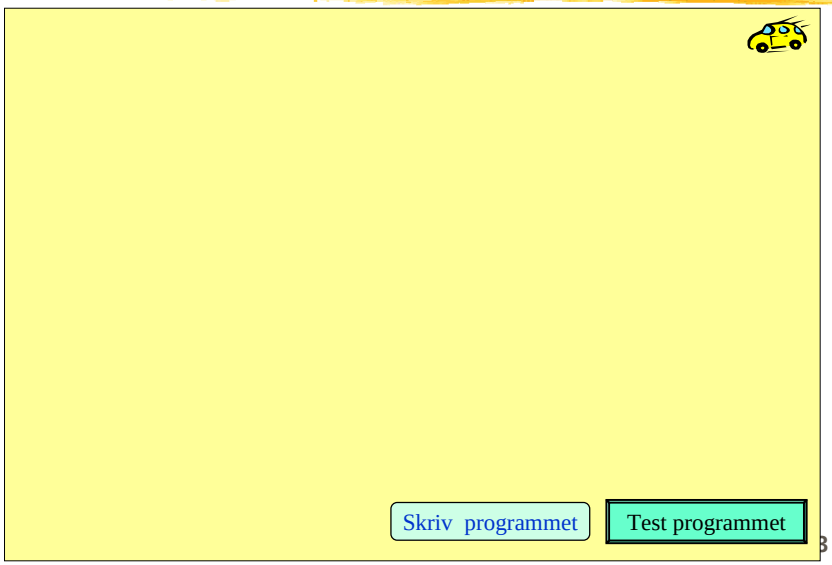
Vi må kjenne formelen for å regne ut BMI. La

vekt = personens vekt i kg
hoyde = personens høyde i m

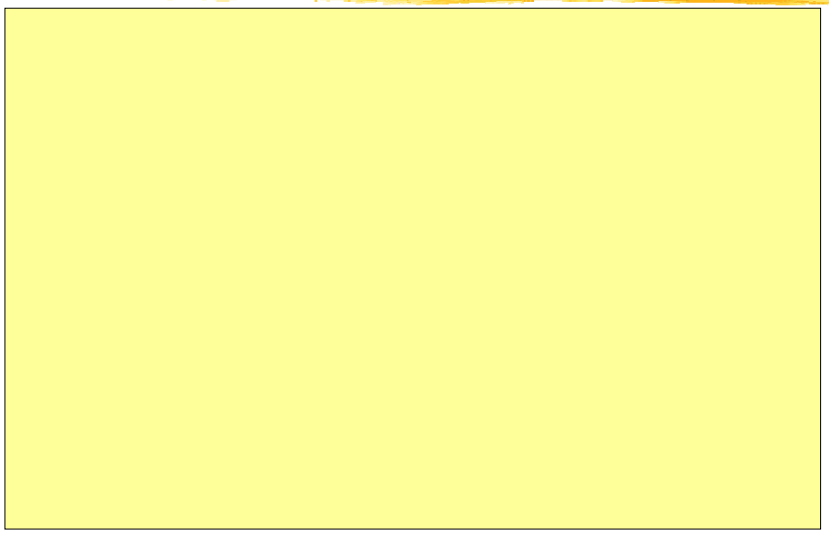
Da er

BMI = vekt / (hoyde*hoyde)

Ferdig program



Ferdig program (*forts.*)



Eksempel: beregne pulssoner

Det har i media vært mye fokus på hvordan man kan sikre at fysisk trening foregår med riktig intensitet. Ett hjelpemiddel i den forbindelse er å beregne sine egne pulssoner og sørge for å ligge i riktig sone når man trener.

Sone	Puls fra	Puls til	Forklaring
1	95	114	50-60%
2	114	133	60-70%
3	133	152	70-80%
4	152	171	80-90%
5	171	190	90-100%

aerob trening

anaerob trening


```
import easyIO.*;

class Pulssoner {
  public static void main (String[] args) {
    In tast = new In();
    Out skjerm = new Out();

    System.out.print("Oppgi hvilepuls: ");
    int hvilepuls = tast.inInt();

    System.out.print("Oppgi makspuls: ");
    int maxpuls = tast.inInt();
```



```
skjerm.out("Sone", 5, Out.RIGHT);
skjerm.out("Min", 5, Out.RIGHT);
skjerm.out("Max", 5, Out.RIGHT);
skjerm.outln("  Beskrivelse");
```



```
  int sone = 1;
  String beskrivelse = "50-60% av maksimum";
  int fra = (int) (hvilepuls + (maxpuls-hvilepuls)*0.50);
  int til = (int) (hvilepuls + (maxpuls-hvilepuls)*0.60);
  skjerm.out(sone, 5);
  skjerm.out(fra, 5);
  skjerm.out(til, 5);
  skjerm.outln("  " + beskrivelse);

  ... tilsvarende for sone 2, 3, 4 og 5 .....

 }  // Slutt på main-metoden

}  // Slutt på class Pulssoner
```

Test programmet

Alternativ til if-else: switch

- En sammensetning av flere if-setninger kan i noen tilfeller erstattes med en switch-setning:

Et uttrykk som gir en verdi som er av en av typene char eller int (evt. byte eller short)

```
switch (uttrykk) {
   case verdi1:
      <instruksjoner>
      break;
   ....
   case verdiN:
      <instruksjoner>
      break;
   default:
      <instruksjoner>
}
```

- Nøkkelordet break avbryter utførelsen av switch-setningen. Når break mangler, fortsetter utførelsen på neste linje (det er sjelden ønskelig).

Eksempel 1

```
class BrukAvSwitch {
  public static void main (String [] args) {
    char c = 'b';

    switch(c) {
      case 'a':
        System.out.println("Tegnet var en a");
        break;
      case 'b':
        System.out.println("Tegnet var en b");
        break;
      default :
        System.out.println("Tegnet var ikke a eller b");
    }
  }
}
```

Endre programmet | Test programmet

Eksempel 2

Dette virker **ikke** (hvorfor?):

```
class FeilBrukAvSwitch {
  public static void main (String [] args) {
    String s = "hovedstad";

    switch(s) {
      case "Paris":
        System.out.println("Vi er i Frankrike");
        break;
      case "Moskva":
        System.out.println("Vi er i Russland");
        break;
      case "Kathmandu" :
        System.out.println("Vi er i Nepal");
    }
  }
}
```


Oppgave 1

- Hva blir skrevet ut av dette programmet ?

```
class IfTest {
   public static void main (String [] args) {
     String s = "Petter";

     if (s.equals("Jens"));
     {
       System.out.println("Ordet var " + s);
     }
   }
}
```

Test programmet

Oppgave 2

- Hva blir skrevet ut av dette programmet?

```
class IfTest2 {
   public static void main (String [] args) {
     double x = -0.5;
     double y = 0.5;

     if (Math.ceil(x) == Math.ceil(y)) {
       System.out.println("A");
     }
     if ((int) x == (int) y) {
       System.out.println("B");
     }
     if (x < y) {
       if (x < 0) {
         if (y < 0) {
           System.out.println("C");
         }
       } else {
         System.out.println("D");
       }
     }
   }
}
```

Test programmet

Oppgave 3

- Hva blir skrevet ut av dette programmet?

```
class Divisjon {
   public static void main (String [] args) {
     if (1/2 > 0) {
       System.out.println("A");
     } else {
       System.out.println("B");
     }
   }
}
```

Test programmet

Oppgave 4

- Hva blir skrevet ut av dette programmet?

```
class TestSwitch {
   public static void main (String [] args) {
     char svar = '@';

     switch(svar) {
       case '@':
         System.out.println("Svaret er en alfakrøll");
       case '!':
         System.out.println("Svaret er et utropstegn");
       default :
         System.out.println("Svaret er verken @ eller !");
     }
   }
}
```

Test programmet

Litt mer om uttrykk: ++ og --

Instruksjon	Alternativ 1 Prefiks-operator	Alternativ 2 Postfiks-operator
`i = i + 1`	`++i`	`i++`
`i = i - 1`	`--i`	`i--`

- `++i, i++, --i og i--` endrer ikke bare på verdien til i, de er dessuten uttrykk som selv har en verdi. Dermed kan vi f.eks. skrive:

```
System.out.println(i++);  // Skriv ut i og øk deretter i med 1
System.out.println(++i);  // Øk i med 1 og skriv deretter ut i
```

- Prefiks-operatorene endrer verdien til variabelen før uttrykket er evaluert.
- Postfiks-operatorene endrer verdien etter at uttrykket er evaluert.

Oppgave

- Fyll ut de tomme feltene i tabellen:

Programkode	Verdien til k	Verdien til m	Verdien til n

`int k = 0;`
`int m;`
`int n;`
	0	udefinert	udefinert

`k = k + 1;`
`m = ++k;`
`n = k++;`
	3	2	2

```
k = k + 1;
k = k + 1;
m = k;
n = k;
k = k + 1;
```


Blokker

- En blokk er en samling instruksjoner omgitt av krøllparenteser:

```
{
 instruksjon 1;
 instruksjon 2;
 ....
 instruksjon n;
}
```

- Alle steder i et Java-program hvor det kan stå en instruksjon, kan vi om ønskelig i stedet sette en blokk.

Deklarasjoner inne i blokker

- Vi har lov til å deklarere variabler inne i en blokk, forutsatt at de ikke allerede er deklarert utenfor blokken. Eksempel:

```
double x = 0.3;
if (x < 0) {
 double y;      // Her er y deklarert inne i en blokk
 y = -x;
}
```

- Variabler deklarert inne i en blokk eksisterer ikke utenfor blokken.

```
double x = 0.3;
if (x < 0) {
 double y;
 y = -x;
}
x = y;     // Ulovlig, siden y ikke eksisterer her
```


while-løkker

- Vi kan utføre en instruksjon/blokk flere ganger ved hjelp av en while-løkke:

```
while (logisk uttrykk) {
  setning 1;
  setning 2;
  .....
  setning n;
}
```

- Hvis det logiske uttrykket er `true`, utføres setningene i while-løkka.
- Dette gjentas inntil det logiske uttrykket er `false`. Da avsluttes løkka.

Eksempel

```
class SkrivLinjer {
   public static void main (String [] args) {
     int k = 1;

     while (k <= 5) {
       System.out.println("Nå har k verdien " + k);
       k = k + 1;
     }

     System.out.println("Nå er k lik " + k);

   }
}
```


Kompilering og kjøring

```
> javac SkrivLinjer.java
> java SkrivLinjer
Nå har k verdien 1
Nå har k verdien 2
Nå har k verdien 3
Nå har k verdien 4
Nå har k verdien 5
Nå er k lik 6
```


Oppgave

- Hva blir utskriften fra dette programmet?

```
class LokkeTest {
   public static void main (String [] args) {
     int k = 3;

     while (k > 0) {
       System.out.print("Nå er k = ");
       System.out.println(k);
       k = k - 1;
     }
   }
}
```

Test programmet

Oppgave 2

- Hva blir utskriften fra dette programmet?

```
class LokkeTest2 {
   public static void main (String [] args) {
     int i = 1;
     int j = 6;

     while (i < j) {
       System.out.println("i = " + i);
       System.out.println("j = " + j);
       System.out.println();
       i = i + 1;
       j = j - 1;
     }
   }
}
```

Test programmet

i	j

Eksempel: gangetabell

- Problem:

  Lag et program som bruker en while-løkke til å beregne "tregangen" og lage en slik utskrift på skjermen:

```
1 * 3 = 3
2 * 3 = 6
3 * 3 = 9
4 * 3 = 12
5 * 3 = 15
6 * 3 = 18
7 * 3 = 21
8 * 3 = 24
9 * 3 = 27
10 * 3 = 30
```


Løsningsskisse

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

    <deklarasjoner>

    <initialisering av variable>

    <while-løkke, hvor hvert gjennomløp
     regner ut og skriver ut en ny linje
     på skjermen>

  }
}
```


while-løkka

```
while (ikke ferdig) {
   <finn svaret på neste regnestykke>
   <skriv ut svaret>
}
```

- For å finne svaret på neste regnestykke, må vi holde rede på de to tallene som skal ganges sammen. Vi trenger bare en variabel for dette, siden annen faktor alltid er lik 3. Vi lager også en variabel for svaret:

```
int k, svar;
```

- Vi initialiserer slik:

```
k = 1;
```

- Dermed blir svaret på neste regnestykke:

```
svar = k * 3;
```


while-løkka *forts.*

```
while (ikke ferdig) {
   int svar = k * 3;
   System.out.print(k + " * 3 = " + svar);
}
```

- Vi må i tillegg huske å endre verdien til `k` i slutten av hvert gjennomløp (ellers gjør vi samme regnestykke igjen og igjen):

```
k = k + 1;
```

- Hvor lenge skal while-løkka løpe ? Vi kan bruke denne testen:

```
k <= 10
```

Ferdig program

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

    int k = 1;
    while (k <= 10) {
      int svar = k * 3;
      System.out.println(k + " * 3 = " + svar);
      k = k + 1;
    }
  }
}
```


Kompilering og test

```
> javac TreGangen.java
> java TreGangen
1 * 3 = 3
2 * 3 = 6
3 * 3 = 9
4 * 3 = 12
5 * 3 = 15
6 * 3 = 18
7 * 3 = 21
8 * 3 = 24
9 * 3 = 27
10 * 3 = 30
```


Eksempel: innlesning med sjekk

- **Problem:**

  Lag et program som leser et heltall mellom 1 og 100 fra terminal.

  Hvis det innleste tallet ikke ligger i det lovlige intervallet, skal programmet be om nytt tall.

  Dette gjentas inntil brukeren skriver et lovlig tall.

Ferdig program

```
import easyIO.*;

class LesVerdi {
  public static void main (String[] args) {
    In tast = new In();

    System.out.print("Oppgi verdi (1,2,...,100): ");
    int verdi = tast.inInt();

    while (!(verdi >= 1 && verdi <= 100)) {
      System.out.println("Ulovlig verdi!");
      System.out.print("Prøv igjen: ");
      verdi = tast.inInt();
    }

    System.out.println("Du oppga verdien " + verdi);
  }
}
```

Test programmet

Evig løkke

- Dersom testen i while-løkka aldri blir usann (false), vil utførelsen av while-løkka aldri stoppe. Dette kalles en evig løkke.
- To eksempler:

```
class EvigLokke1 {
  public static void main (String [] args) {
    while (true) {
      System.out.println("INF 1000");
    }
  }
}
```

```
class EvigLokke2 {
  public static void main (String [] args) {
    int i = 1, j = 2;
    while (i < j) {
      System.out.println("Nå er i < j");
    }
  }
}
```


Kompilering og kjøring

```
> javac EvigLokke1.java
> java EvigLokke1
INF 1000
INF 1000
INF 1000
INF 1000
INF 1000
INF 1000
   ...
   ...
   ...
 (osv)
```

```
> javac EvigLokke2.java
> java EvigLokke2
Nå er i < j
Nå er i < j
Nå er i < j
Nå er i < j
Nå er i < j
Nå er i < j
   ...
   ...
   ...
 (osv)
```


Variant av while: do-while

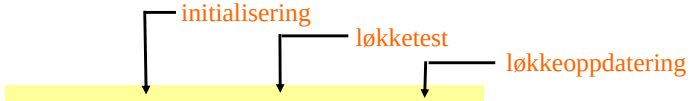
- Formen på en do-while løkke:

```
do {
   <setning 1>
   <setning 2>
   .....
   <setning n>
} while (logisk uttrykk);
```

- Noen foretrekker denne fremfor while-løkker når løkke-innmaten alltid skal utføres minst en gang.

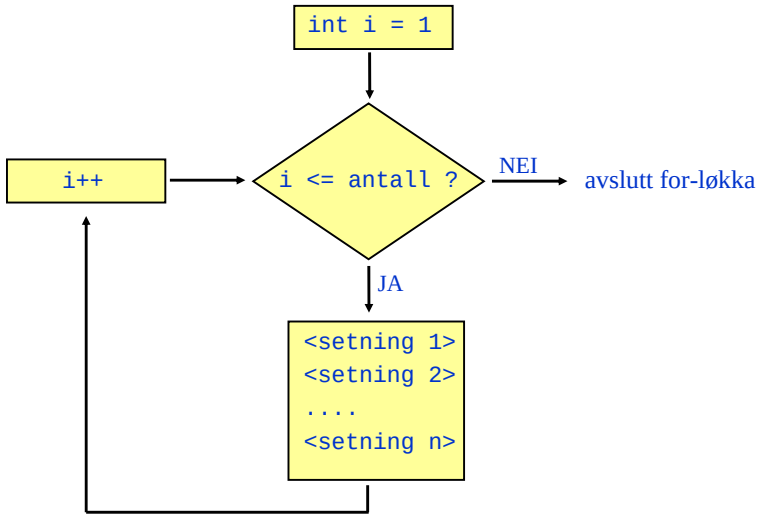
for-løkker

- En annen måte å få utført en instruksjon (eller blokk) mange ganger er ved hjelp av en for-løkke:




```
for (int i=1; i<=antall; i++) {
  <setning 1>
  <setning 2>
  ....
  <setning n>
}
```

Hvordan for-løkka virker



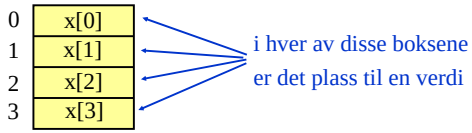
Nesting av løkker

- Det er ofte behov for å neste løkke-setninger inne i hverandre; vi kommer til å se mange eksempler etterhvert.
- Eksempel på nestet for-løkke:

```
for (int i=0; i<10; i++) {
  for (int j=0; j<10; j++) {
    int produkt = i * j;
    System.out.println("i * j = " + produkt);
  }
}
```


Arrayer

- Hittil har vi sett på variable som kan holde en enkelt verdi:
  - en int-variabel har plass til ett heltall
  - en String-variabel har plass til en enkelt tekststreng
  - osv.
- Arrayer er "variable" som kan holde på mange verdier:
  - en int-array har plass til mange heltall
  - en String-array har plass til mange tekststrenger
  - osv.
- Verdiene som ligger i en array har hver sin posisjon (= indeks):
  0, 1, 2, ..., K-1 hvor K = lengden til arrayen
- En array x med lengde 4 kan visualiseres slik:



Deklarere og opprette arrayer

- Deklarere en array (gi den et navn):

```
datatype[] arrayNavn;
```

f.eks. int, double, boolean eller String

- Opprette en array (sette av plass i hukommelsen):

```
arrayNavn = new datatype[K];   // K er ønsket lengde
```

- Deklarere og opprette i en operasjon:

```
datatype[] arrayNavn = new datatype[K];
```

- Eksempler:

```
int[] a = new int[10];
double[] x = new double[100];
String[] s = new String[1000];
```

Verdiene i en array

- Anta at vi har deklarert og opprettet følgende array:

```
int[] tlf = new int[600];
```

- For å få tak i de enkelte verdiene i arrayen:

```
tlf[0], tlf[1], tlf[2], ..., tlf[599]
```

- For å få tak i lengden på arrayen:

```
tlf.length      // NB: ingen parenteser til slutt
```

- For å sortere elementene i en array (i stigende rekkefølge):

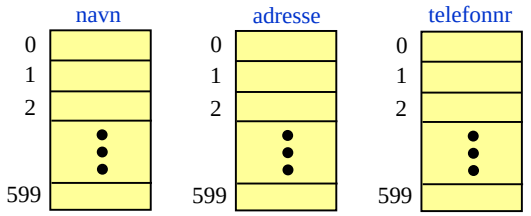
```
java.util.Arrays.sort(tlf);
```


Eksempel på bruk av arrayer

- Anta at vi ønsker å lagre navn, adresse og telefonnr for de som følger et bestemt kurs med maksimalt 600 studenter:

```
String[] navn = new String[600];
String[] adresse = new String[600];
int[] telefonnr = new int[600];
```

- Resultatet kan visualiseres slik:



Eksempel: lese og skrive ut

- Program som leser tre ord fra terminal og skriver dem ut igjen:

```
import easyIO.*;

class LesOgSkriv {
  public static void main (String [] args) {
     In tastatur = new In();
     String[] s = new String[3];

     for (int i=0; i<3; i++) {
        System.out.print("Ord: ");
        s[i] = tastatur.inLine();
     }

     for (int i=0; i<3; i++) {
        System.out.println(s[i]);
     }
  }
}
```

Test programmet

Eksempel: lese og skrive ut sortert

- Program som leser tre navn fra terminal og skriver dem ut i sortert rekkefølge:

```
import easyIO.*;

class LesOgSorter {
  public static void main (String [] args) {
     In tastatur = new In();
     String[] s = new String[3];

     for (int i=0; i<3; i++) {
        System.out.print("Ord: ");
        s[i] = tastatur.inLine();
     }

     java.util.Arrays.sort(s);

     for (int i=0; i<3; i++) {
        System.out.println(s[i]);
     }
  }
}
```

Test programmet

Automatisk initialisering av arrayer

- Når en array blir opprettet, blir den automatisk initialisert (dvs verdiene er ikke udefinerte når den er opprettet).
  - `int[] k = new int[100];` // **Nå er alle k[i] == 0**
  - `double[] x = new double[100];` // **Nå er alle x[i] == 0.0**
  - `boolean[] b = new boolean[100];` // **Nå er alle b[i] == false**
  - `char[] c = new char[100];` // **Nå er alle c[i] == '\u0000'**
  - `String[] s = new String[100];` // **Nå er alle s[i] == null**
- Merk: String-arrayer initialiseres med den spesielle verdien `null`. Dette er *ikke* en tekststreng og må ikke blandes sammen med en tom tekst: `""`.
- For å kunne bruke verdien `s[i]` til noe fornuftig må du først sørge for å gi `s[i]` en tekststreng-verdi, f.eks. `s[i] = "Per"` eller `s[i] = "".`

Egendefinert initialisering av en array

- Det er ikke alltid den automatiske initialiseringen av en array gir det vi ønsker. Vi kan da initialisere arrayen med våre egne verdier, slik som i disse eksemplene:

```
int[] primtall = {2, 3, 5, 7, 11, 13};

double[] halve = {0.0, 0.5, 1.0, 1.5, 2.0};

String[] ukedager = {"Mandag", "Tirsdag", "Onsdag",
      "Torsdag", "Fredag", "Lørdag", "Søndag"};
```