



INF1000 – Forelesning 6

Repetisjon




Oversikt

- Variable
- Uttrykk
- Innlesing fra terminal
- Formatert utskrift til skjerm
- Skop
- Forgrening (`if/switch`)
- Løkker (`while/do/for`)
- Arrayer
- Metoder



Variabel – en plass i lageret

- En plass i maskinens lager (minne) kan ses på som
 - en skuff i en kommode, eller som
 - en biloppstillingsplass på en parkeringsplass
- De kan ha forskjellige størrelser avhengig størrelsen på det dataelementet vi skal legge der
- Variable må ha **navn**
 - Slik at vi kan angi i programmet vårt hvilken plass i lageret vi snakker om
- Variable må ha **type**
 - Så vi vet hvor stor plass variabelen tar og hva det er som ligger der



Variable – Deklarasjon

- Deklarasjon
 - Angi navn og type til en variabel
 - Er en setning i programmet

`int radius;`

Type

Navn

- Kan deklarere flere variable i samme setning:


```
int radius, diamanter, omkrets;
```

Variable – Primitive typer

Type	Forklaring	Eksempel
byte	heltall	<code>byte b = 9;</code>
short	heltall	<code>short s = 256;</code>
int	heltall	<code>int i = 76234</code>
long	heltall	<code>long l = 12345690L</code>
float	desimaltall	<code>float f = 2.5F</code>
double	desimaltall	<code>double d = 3.14</code>
char	ett tegn	<code>char c = '9';</code>
boolean	sannhetsverdi	<code>boolean b = true;</code>

Variable - String

- Brukes ofte på samme måte som de primitive, men er ikke en primitiv type
- Skrives inne i ""
 - For eksempel: `"Fredrik Sørensen"`, `"INF1000"`
- String-variable deklarerer slik:

```
String navn;
```

Variable – Tilordning

- Variable har ingen verdi i utgangspunktet
- Verdien settes med en tilordningssetning
 - Anta at `radius` er av type `double`:
`radius = 4.65;`
- En variabel kan tilordnes flere ganger
 - Den eksisterende verdien vil da overskrives
- En variabel kan tilordnes hvor som helst etter at den er deklart
- NB: `=` betyr **tilordning** i Java, må ikke forveksles med matematisk likhet!

Variable – Tilordning og avlesing

- Opprette en lagerplass i maskinen til et heltall:

```
int lengde;  
int svar;
```

- Fylle plassen med en verdi (et heltall):

```
lengde = 38;
```

- Avlese (eller bruke) verdien:

```
svar = lengde * 2;
```

Tilordner verdien til variabelen

Variable – Regne ut areal

```
public class Areal {

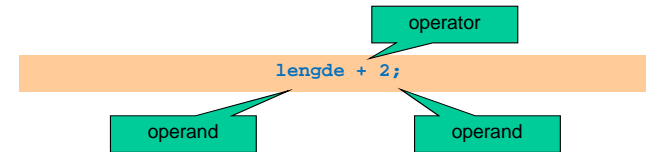
    public static void main(String[] args){
        double pi = 3.14;
        double radius = 2.0;

        // Regner ut arealet og skriver ut resultatet
        double areal = pi * radius * radius;
        System.out.println("Areal av en sirkel med radius " +
            radius + " er " + areal);

        // Øker arealet med 2 og gjentar oppgaven.
        radius = radius + 2;
        areal = pi * radius * radius;
        System.out.println("Areal av en sirkel med radius " +
            radius + " er " + areal);
    }
}
```

Uttrykk

- Operatorer
 - Utføres på en eller flere operander



- Metodekall
 - Operandene forekommer inne i parenteser

```
Math.ceil(x);
```

Uttrykk

- Aritmetiske uttrykk:


```
2 * (3+4) / 1.5
2 / (12 + 34 - 2.3)
```
- Logiske uttrykk:

Uttrykket	har verdien true hvisog verdien false ellers
<code>x < y</code>	<code>x</code> mindre enn <code>y</code>	
<code>x <= y</code>	<code>x</code> mindre enn eller lik <code>y</code>	
<code>x == y</code>	<code>x</code> lik <code>y</code>	
<code>x != y</code>	<code>x</code> ikke lik <code>y</code>	
<code>x > y</code>	<code>x</code> større enn <code>y</code>	
<code>x >= y</code>	<code>x</code> større enn eller lik <code>y</code>	
<code>! (x < y)</code>	ikke <code>x</code> mindre enn <code>y</code>	
<code>b1 && b2</code>	både <code>b1</code> og <code>b2</code> sann	
<code>b1 b2</code>	<code>b1</code> eller <code>b2</code> (eller begge) sann	

Tolking av literaler

- Tallene vi skriver i programmet vårt kalles literaler.
- De som kun inneholder tall tolkes som **int**.

```
int verdi = 12345;
```

- De som inneholder desimaltegn tolkes som **double**.

```
double verdi = 12.345;
```

Innlesning fra terminal

- Innlesning fra terminal kan gjøres på flere måter i Java. I INF1000 bruker vi pakken easyIO. Du må da skrive i toppen av programmet:


```
import easyIO.*;
```
- Inne i klassen skriver vi følgende før vi kan starte innlesning:


```
In tastatur = new In();
```
- Så kan vi lese inn fra terminal (=tastatur), f.eks. et heltall:


```
int radius;
System.out.print("Oppgi radiusen: ");
radius = tastatur.inInt();
```

Eksempel

```
import easyIO.*;

class LesFraTerminal {
    public static void main (String [] args) {
        In tast = new In();
        System.out.print("Skriv et heltall: ");

        int k = tast.inInt();

        System.out.println("Du skrev: " + k);
    }
}
```

Vi importerer pakken easyIO.

Vi oppretter en verktøykasse for lesing fra terminal og lager en variabel tast som blir vårt håndtak til denne verktøykassen.

I verktøykassen ligger det bl.a. en metode for å lese et heltall fra terminalen.

Hvordan lesemetodene virker

Terminal-input: `__ x y z _ 1 6 1 2 7 5` `_` = blank

<pre>String s1 = tast.inWord(); String s2 = tast.inWord();</pre>	→	<pre>s1: "xyz" s2: "161275"</pre>
<pre>String s1 = tast.inWord(); int x = tast.inInt();</pre>	→	<pre>s1: "xyz" x : 161275</pre>
<pre>String s = tast.inLine();</pre>	→	<pre>s: " xyz 161275"</pre>
<pre>char c1 = tast.inChar(); char c2 = tast.inChar(); char c3 = tast.inChar();</pre>	→	<pre>c1: ' ' c2: ' ' c3: 'x'</pre>
<pre>int x = tast.inInt();</pre>	→	<pre>feilmelding</pre>

Formatert utskrift til skjerm

- Formatert utskrift vil si at vi angir nøyaktig hvordan utskriften skal se ut og plasseres på skjermen.
 - Kan gjøres "manuelt" med System.out.print(...), men det er upraktisk.
- Bedre: bruke en ferdiglaget pakke for slikt. I INF1000 bruker vi pakken easyIO. I toppen av programmet (**før** class) skriv:


```
import easyIO.*;
```
- Inne i klassen skriver vi så:


```
Out skjerm = new Out();
```
- Så kan vi skrive ut det vi ønsker, f.eks.:


```
double pi = 3.1415926;
skjerm.out(pi, 2, 6); // Skriv ut pi med 2 desimaler
// høyrejustert på 6 plasser.
```

Blokker

- En **programblokk** er en samling med programsetninger omsluttet av krøllparenteser
- Setningene i main-metoden ligger inne i en blokk
- Blokker kan **nøstes** inne i hverandre, slik at vi kan ha blokker inne i blokker
- En variabel som er deklartert inne i en blokk er kun definert ("synlig") fra stedet den er deklartert til slutten av blokken. Vi kaller det **skopet** til variabelen.

Skop – Lovlig eksempel

```
class SkopLovlig {
    public static void main(String args[]){
        int k = 15;
        {
            int n = 10;
            System.out.println(k + n);
        }
        // Her er ikke n definert
        System.out.println(k);
    }
}
```

Skop – Ikke lovlig eksempel

```
class SkopIkkeLovlig {
    public static void main(String args[]){
        int k = 15;
        {
            int n = 10;
            int k = 200; // Ikke lov.
                                // k er allerede
                                // definert.
        }
    }
}
```

Programmer med forgreninger

- En svært nyttig programmeringsteknikk er å bruke forgreninger, dvs forskjellige instruksjoner utføres i ulike situasjoner.
- Vi kan få til dette med en **if-setning** (pseudokode):

```
if (logisk uttrykk)
{
    <instruksjoner>
}
else
{
    <instruksjoner>
}
```

et uttrykk som enten er true eller false, f.eks. $x < y$

Den første blokken (og bare den) blir utført hvis det logiske uttrykket er sant (true)

Den andre blokken (og bare den) blir utført hvis det logiske uttrykket er usant (false)

- Eksempel:

```
if (x > 0) {
    System.out.println("Tallet er positivt");
} else {
    System.out.println("Tallet er ikke positivt");
}
```

Varianter av if-setninger

- Else-delen kan utelates:

```
if (pris > 1500) {System.out.println("For dyrt"); }
```
- Vi kan legge if-setninger inni if-setninger:

```
if (lønn < 400000) {
  if (ferieuker < 8) {
    System.out.println("Ikke søk på jobben");
  }
}
```
- Vi kan lage sammensatte if-setninger:

```
if (a < 10) { // a ikke er positivt heltall
  System.out.println("Ett siffer");
} else if (a < 100) {
  System.out.println("To siffer");
} else {
  System.out.println("Mer enn to siffer");
}
```

Eksempel på bruk av if-setning

Program som avgjør hvem av to personer som er høyest:

```
import easyIO.*;
class Hoyde {
  public static void main (String[] args) {
    In tastatur = new In();
    double høyde1, høyde2;
    System.out.print("Høyden til Per: ");
    høyde1 = tastatur.inDouble();
    System.out.print("Høyden til Kari: ");
    høyde2 = tastatur.inDouble();

    if (høyde1 > høyde2) {
      System.out.println("Per er høyere enn Kari");
    } else {
      System.out.println("Per er ikke høyere enn Kari");
    }
  }
}
```

Alternativ til if-else: switch

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

```
switch (uttrykk) {
  case verdil:
    <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

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

while-løkker



- Vi kan utføre en blokk med setninger flere ganger ved hjelp av en while-løkke

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

- Hvis det logiske uttrykket er sant, utføres setningene i while-løkka.
- Dette gjentas inntil det logiske uttrykket er usant. Da avsluttes løkka.

25

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);
    }
}
```

26

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.

27

for-løkker



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

```
for (<initialisering>; <betingelse>; <oppdatering>){
    <setning 1;>
    <setning 2;>
    ....
    <setning n;>
}
```

28

Eksempel på for-løkke

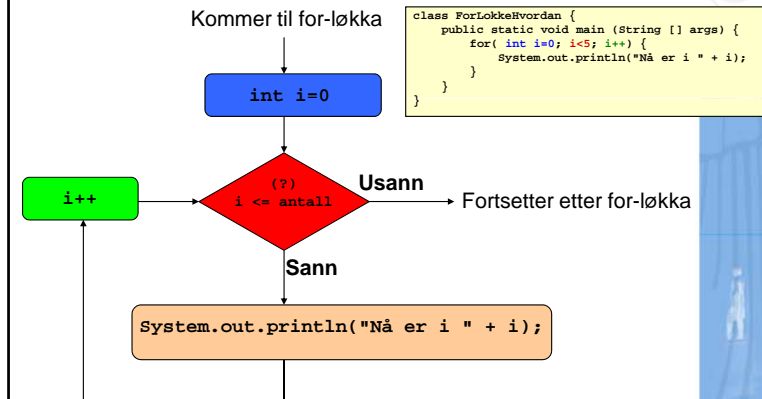


```
class ForLokkeHvordan {
    public static void main (String [] args) {
        for( int i=0; i<5; i++) {
            System.out.println("Nå er i " + i);
        }
    }
}
```

Initialisering
Oppdatering
Betingelse

```
$ java ForLokkeHvordan
Nå er i 0
Nå er i 1
Nå er i 2
Nå er i 3
Nå er i 4
$
```

Hvordan for-løkka virker

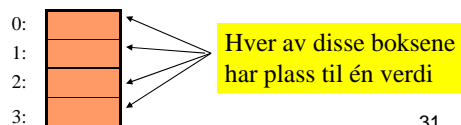


30

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, ..., N-1 hvor N = lengden til arrayen
- En array x med lengde 4 kan tegnes slik:



31

Deklarere og opprette arrayer



- Deklarere en array (gi den et navn):
`<datatype>[] arrayNavn;`
- Opprette en array (sette av plass i hukommelsen):
`arrayNavn = new <datatype>[K];`
- 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];`

32

Verdiene i en array



- Anta at vi har deklartert 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
```

33

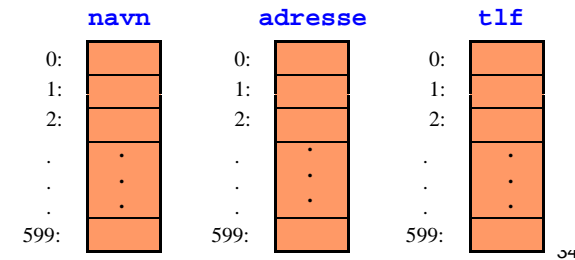
Eksempel på bruk av arrayer



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

- Resultatet kan visualiseres (tegnes) slik



Eksempel: lese og skrive ut



- Program som leser data om et antall personer fra input.

```
import easyIO.*;
class LesInnPersoner {
    public static void main (String [] args) {
        In tast = new In();
        String[] navn = new String[3];
        for (int i=0; i<navn.length; i++) {
            System.out.print("Navn: ");
            navn[i] = tast.inLine();
        }
        for (int i=0; i<navn.length; i++) {
            System.out.println(navn[i]);
        }
    }
}
```

Oppretter array

Legger inn verdi

Bruker lengden i betingelsen

Leser ut verdi

35

Automatisk initialisering av arrayer



- Når en array blir opprettet, blir den automatisk initialisert (dvs verdiene er ikke udefinerte når arrayen 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] = ""`.
- Generelt, når vi bruker `new`, får vi 'null-fylt' det vi lager med `new`. (mye mer bruk av `new` senere)

36

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

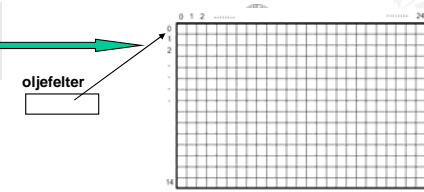
```
double[] halve = {0.0, 0.5, 1.0, 1.5, 2.0};
```

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

37

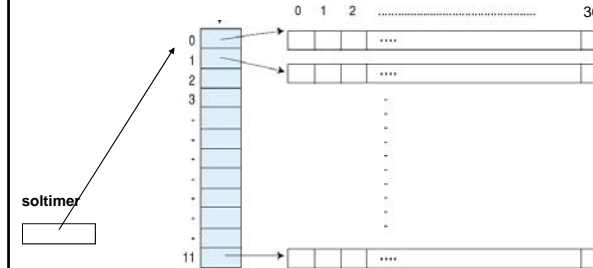
To-dimensjonale (2D) arrayer

- slik tenker vi oss det
- og slik er det



```
int[][] soltimer = new int[12][31];
```

gir følgende:



Figur 5.3 En todimensjonal array med arraypekere og arrayobjekter.

Blokker og metoder

- **Blokk:** samling setninger omgitt av krøllparenteser:

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

- Enhver **setning** i et Java-program kan erstattes med en **blokk**
- En blokk forekommer ofte flere steder i et program:
 - Definere blokken *ett* sted og gi den et navn
 - Angi navnet hvert sted vi ønsket setningene i blokken utført
- I Java kalles dette en **metode**
- Skiller mellom
 - **deklarerer** (lage/skrive) en metode
 - **kalle** (bruke) en metode

39

Metode-deklarasjon

- Generelt har en metode-deklarasjon følgende form:

```

modifikatorer returverditype metodenavn (parametre) {
    setning 1;
    setning 2;
    ....
    setning n;
}

```

Annotations: "mer om denne senere" points to "modifikatorer". "beskrivelse av hva slags output metoden gir, f.eks. void, int, double, char, ..." points to "returverditype". "et navn som vi velger" points to "metodenavn". "beskrivelse av hva slags input metoden skal ha - gis i form av variabel-deklarasjoner separert av komma" points to "parametre".

Merk: en metode kan kreve input og den kan returnere en verdi, men ingen av delene er nødvendig. I enkleste tilfelle er det ingen input og ingen output.

Eksempel 2: ingen parametre

- Følgende metode skriver ut fire linjer med stjerner på skjermen:

```
void skrivStjerner () {
    String s = "*****";
    System.out.println(s);
    System.out.println(s);
    System.out.println(s);
    System.out.println(s);
}
```

- `skrivStjerner` er metodens navn
- Returverditype `void` forteller at metoden ikke gir noen returverdi

41

Eksempel 3: returverdi og parametre

- Følgende metode summerer to tall og returnerer resultatet:

```
int summer(int tall1, int tall2) {
    int sum;
    sum = tall1 + tall2;
    return sum;
}
```

- `summer` er metodens navn
- Returverditype er `int`
 - Metoden returnerer en verdi av type `int`
 - Gjøres med setningen `return int;`
- Metoden har to parametre
 - `tall1` av type `int`
 - `tall2` av type `int`

42

Kalle på en metode

- Kalle på metode:** benytte/utføre metoden
- Kalle på en metode uten parametre:


```
metodenavn();
```
- Kalle på en metode med parametre:
 - Må oppgi like mange verdier som metoden har parametre
 - Verdiene må ha samme datatype som parameterne i metode-deklarasjonen
 - Eksempel:


```
metodenavn2(34.2, 53, 6);
```
- Hvis metoden returnerer en verdi:
 - Returverdi *kan* tas vare på:


```
int alder = metodenavn3(25.3, 52, 7);
```

43

Metoder vs. klasser

- Metoder:** *handlingene* i programmet
 - Består av vanlige programsetninger
 - Gis et navn (som vi velger selv)
 - Lurt: gi metoden et navn som beskriver det den gjør
 - Banksystem: En metode for hver av handlingene *innskudd, uttak, beregnRenter, skrivRapport, ...*
- Klasser:** data og metoder som hører naturlig sammen
 - Programmer deles i klasser tilsvarende en naturlig oppdeling av problemet
 - Banksystem: En klasse for hver av *Banken, Kunde, Konto, ...*

Huskeregel: En klasse **er** noe, en metode **gjør** noe.
Metodene er inne i klasser.

Metoder i to klasser skal vi lære idag

Klasser, objekter og metoder i flere klasser skal vi lære senere

Fra nå: Nytt oppsett for programmer

```
import easyIO.*;

class MittProgram {
    public static void main (String[] args) {
        // her lager vi et objekt av den andre klassen
        Student s = new Student();
        // her kan vi kall på metodene i Student - eks:
        s.skriv();
        System.out.println("Programmet ferdig - ha det");
    }
}

class Student {
    String navn; // evt. data i klassen 'Student'
    Student() {
        // startmetode, f.eks initialisering
        navn = "Ola";
    }
    void skriv() { // her er en egen metode
        System.out.println("Navnet mitt er:" + navn);
    }
}
```

Levetiden til parametre og variable

- Vi kan ha adgang til tre typer variable i en metode:
 - **Objektvariable:** deklareret på klassenivå, inne i klassen, men utenfor metodene
 - **Lokale variable:** deklarerer inne i metoden, og er definert fra og med der deklarasjonen gjøres og til slutten av blokken de er deklartert i
 - **Parametre:** deklarerer i hodet på metoden, og er definert i hele metodekroppen
- Viktig: ved gjentatte kall på en metode er det et *nytt sett med lokale variable og parametre* som lages hver gang (men det er de samme objektvariablene hvis metoden er i 46 samme objekt).

Parametre og argumenter

```
class Eksempel {
    public static void main (String[] args) {
        A aa = new A();
        aa.minMetode(3.14, 365);
    }
}

class A {
    void minMetode (double x, int y) {
        .....
    }
}
```

Argumenter

Parametre

Merk: et annet navn for argumenter er **aktuelle parametre**, og et annet navn for parametre er **formelle parametre**.

47

Verdien til parameterne kopieres over til metoden

- Ved metodekall overføres verdien til argumentene til metodens parametre slik:

```
public static void main (String[] args) {
    A aa = new A();
    int i = 17;
    aa.minMetode(3.14, i + 2);
}

class A {
    void minMetode (double x, int y) {
        // nå kan x og y brukes med de verdier de
        // fikk i kallet
    }
}
```

x = 3.14;
y = i+2;

Verdiene til argumentene som brukes ved kallet, **blir kopiert over i parameterne før setningene i metoden blir utført.**

48

Metoder med returverdi

```
import easyIO.*;
class Tall {
    public static void main (String[] args) {
        Out skjerm = new Out();
        Leser l = new Leser();
        double x = l.lesPositivtTall();
        double y = l.lesPositivtTall();
        skjerm.out("ln(x*y) = ");
        skjerm.outln(Math.log(x*y), 2);
    }
}
class Leser {
    double lesPositivtTall () {
        In tastatur = new In();
        double x;
        do {
            System.out.print("Gi et positivt tall: ");
            x = tastatur.inDouble();
        } while (x <= 0);
        return x;
    }
}
```

```
> java PositivtTall
Gi et positivt tall: 3.3
Gi et positivt tall: 5.5
ln(x*y) = 2.90
```

Bruk av arrayreferanser som parametre

- I forrige eksempel var parameteren til finnSum en arrayreferanse.
- Det lages ikke noen kopi av arrayobjektet når metoden kalles, så endringer som gjøres på arrayen inni metoden blir synlige utenfor metoden. Hva skriver programmet under ut?

```
class ArrayParameter {
    public static void main (String[] args) {
        int[] a = {1, 2, 3, 4};
        Finn f = new Finn();
        f.finnDelsummer(a);
        System.out.println("a[3] = " + a[3]);
    }
}
class Finn {
    void finnDelsummer(int[] x) {
        for (int i=1; i<x.length; i++) {
            x[i] += x[i-1];
        }
    }
}
```

```
a[3] = 10
```

Overlasting av metoder

- Flere metoder kan deklarerer med samme metodenavn, forutsatt at Java klarer å avgjøre hvilken metode som skal kalles. Krav:
 - metodene har ulikt antall parametre eller
 - metodene har ulik type på noen av parametrene, og slik at Java alltid klarer å finne en entydig match
- Metoden (eller metodenavnet) sies da å være overlastet, og de ulike metodene med samme navn kan ha ulik returtype.
- Eksempel:

```
int sum (int x, int y) {
    return x + y;
}
```

```
double sum (double x, double y) {
    return x + y;
}
```

51

Oppgave 2: hva blir utskriften?

```
class Oppgave2 {
    public static void main (String[] args) {
        GTest gt = new GTest(1);
    }
}
class GTest {
    GTest(int i) {
        while (g(i) > 0) {
            System.out.println(i);
            i = i+1;
        }
    }
    int g (int x) {
        return 5-x;
    }
}
```

```
> javac Oppgave2.java
> java Oppgave2
```

```
1
2
3
4
```

52

Et litt større eksempel



```
class SkrivUt3 {
    public static void main(String[] args) {
        Skriv2 sk = new Skriv2( );
        System.out.println("Her er A i main-metoden");
        sk.skrivMer();
        System.out.println("Her er B i main-metoden");
    }
}

class Skriv2 {
    int k=0;

    int treGanger(int i) {
        int m = k * i * 3;
        return m;
    }

    void skrivMer() {
        k = 4;
        System.out.println("skrivMer kaller treGanger: " + treGanger(2));
    }
}
```

```
> javac SkrivUt3.java
> Java SkrivUt3
Her er A i main-metoden
skrivMer kaller treGanger: 24
Her er B i main-metoden
```



Enklere å løse Oblig2 med metoder



```
import easyIO.*;

class Oblig2 {

    public static void main (String[] args) {
        Olje ol = new Olje();
        ol.ordreløkke();
        System.println("--Avslutter programmet ----");
    } //end main
} // end class Oblig2

class Olje {
    In tast = new In();

    void ordreløkke() {
        int ordre = 0;

        while (ordre != 8) {
            skrivMeny();
            ordre = velgOperasjon();

            switch (ordre) {
                case 1: kjøpFelt(); break;
                case 2: annullerKjøpAvFelt(); break;
                case 3: lagKart(); break;
                case 4: lagSelskapsoversikt(); break;
                case 5: oppdaterOljeutvinning(); break;
                case 6: finnMaksUtvinning(); break;
                case 7: listeFeltUtenOljeutvinning(); break;
                default: break;
            } // end switch
        } // end while flere kommandoer

        System.out.println("**AVSLUTNING PÅ RURITANIAS OLJEFELTSYSTEM**");
    } // end ordreløkke

    // her deklarerer vi de 9 metodene | .....
} // end class Olje
```