INF1000 – Uke 5

Litt mer om arrayer
Metoder

Oversikt

- Svar på spm. fra forrige uke
- Litt mer om arrayer
  - Lete i arrayer
  - Flere dimensjoner
- Metoder
- Neste uke: repetisjon!

Hva er for-løkker - repetisjon

- En 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;>
}
```

Instruksjonene i <oppdatering> utføres, men verdiene forkastes.

Spørsmål forrige uke

- Er det forskjell på disse to for-løkkene?

```
for (int i=0; i<k; i++){ … }

for (int i=0; i<k; ++i){ … }
```

  - Eksempel:

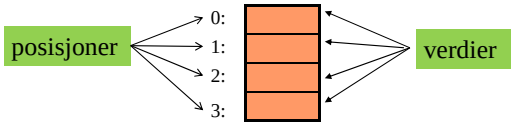
```
int k = 5;

for (int i=0; i<k; i++){ System.out.println(i);

for (int i=0; i<k; ++i){ System.out.println(i);
```

  - Hva skrives ut på skjermen?

```
0
1
2
3
4
0
1
2
3
4
```

Arrayer - repetisjon

- Arrayer er "variable" som kan holde på mange verdier:
  - en int-array har plass til flere heltall
  - en String-array har plass til flere tekststrenger
  - osv.
- Verdiene i en array med lengde n har hver sin posisjon/indeks: 0, 1, 2, . . . , n-1
- En array x med lengde 4 kan tegnes slik:



Eksempel – Finne den yngste

```
import easyIO.*;
class FinnDenYngste {
   public static void main (String [] args) {
      In tast = new In();
      System.out.print("Hvor mange personer? ");
      int antall = tast.inInt();

      String[] navn = new String[antall];
      int[] alder = new int[antall];

      for (int i=0; i<antall; i++) {
         System.out.print("Navn: ");
         navn[i] = tast.inLine();
         System.out.print("Alder: ");
         alder[i] = tast.inInt();
      }

// . . .
```

Leser inn navn og alder i disse.

Eksempel – fortsetter

```
// . . .
      int minste = alder[0];
      int minPos = 0;

      for (int i=1; i<antall; i++) {
         if (alder[i] < minste) {
            minste = alder[i];
            minPos = i;
         }
      }

      System.out.println("Den yngste er " +
            navn[minPos] + " som er " +
            minste + " år");
      }
   }
}
```

Skal hele tiden legge den minste her. Starter med den første

Posisjonen til den minste i arrayen

Sjekker om vi har funnet en som er mindre og oppdaterer i så fall verdiene.

Eksempel – Finne den yngste

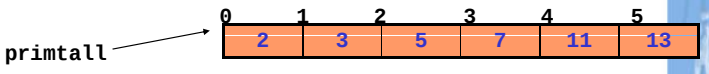
```
$ javac FinnDenYngste.java
$ java FinnDenYngste
Hvor mange personer? 2
Navn: Arild
Alder: 40
Navn: Christian
Alder: 30
Den yngste er Christian som er 30 år
$
```

En array-variabel er en adresse (en peker)

- Når vi deklarerer en array så refererer arraynavnet ikke til selve verdiene i arrayen, men til adressen (i lageret) hvor verdiene ligger lagret.
- Resultatet etter at vi har utført

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

- kan visualiseres slik:



Oppgave

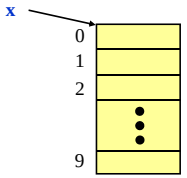
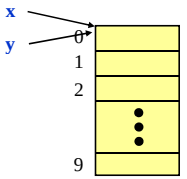
- Hva skriver programmet ut?

```
class ToArrayer {
   public static void main (String [] args) {
      int[] x = new int[5];
      int[] y = x;

      for (int i=0; i<x.length; i++) {
         x[i] = 10 + i;
      }

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


Resultat

```
$ javac ToArrayer.java
$ java ToArrayer
10
11
12
13
14
$
```

Hva skjedde?

Etter å ha utført instruksjonen	.... så er situasjonen denne:
`int[] x = new int[10];`	
`int[] y = x;`	

Kopiering av arrayer

- Vi kan ikke lage en kopi av en array x ved å skrive
  `int[] y = x;`
  siden dette bare medfører at adressen til arrayen legges inn i y.
- Skal vi lage en kopi, må vi først opprette en array til (f.eks. y), og så kopiere over verdiene en for en:

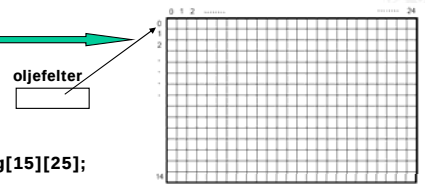
```
double[] y = new double[x.length];

for (int i=0; i<x.length; i++) {
    y[i] = x[i];
}
```


Todimensjonale arrayer

- Vi kan også deklarere todimensjonale (og høyere-dimensjonale) arrayer.
- Eksempel:

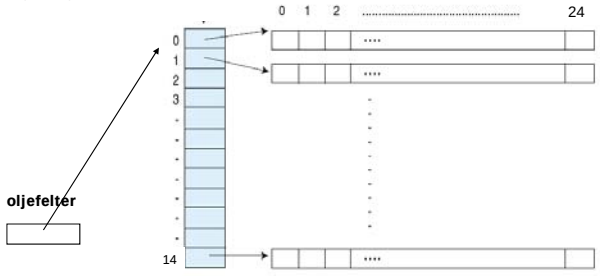
```
String[][] oljefelter = new String[15][25];
```


Todimensjonale arrayer
- slik tenker vi oss det
- og slik er det



String[][] oljefelter = new String[15][25];

gir følgende:



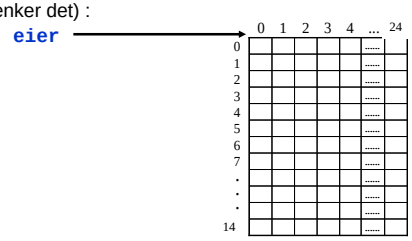
Figur 5.3 En todimensjonal array med arraypekere og arrayobjekter.

Flerdimensjonale arrayer

Eksempel:

```
String[][] eier = new String[15][25];
```

- Resultat (slik vi tenker det) :



- Eksempler på lovlige operasjoner:

```
eier[3][4] = "Petrol A/S";
int antallRader = eier.length;     // 15
int antallKolonner = eier[0].length; // 25
```

Metoder

Motivasjon

- Anta at vi ønsker å sortere arrayen
  `int[] tall = { 2, 1, 4, 3 };`

```
int tmp;

tmp = tall[0];
tall[0] = tall[1];
tall[1] = tmp;

tmp = tall[2];
tall[2] = tall[3];
tall[3] = tmp;
```

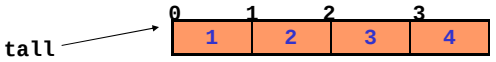
Problem: (nesten) lik kode repeteres!

Løsning: lage en **metode** som bytter om to plasser i arrayen

```
bytt(int a, int b){
   int tmp = tall[a];
   tall[a] = tall[b];
   tall[b] = tmp;
}

bytt(0, 1);
bytt(2, 3);
```

metode

metode-kall



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
  - deklarere (lage/skrive) en metode
  - kalle (bruke) en metode

Metode-deklarasjon (lage metoden)

- Metode: navngitt blokk med setninger
- Metode-deklarasjon:
  - Metodens navn
  - Setningene som skal utføres
- Eksempel: main-metoden

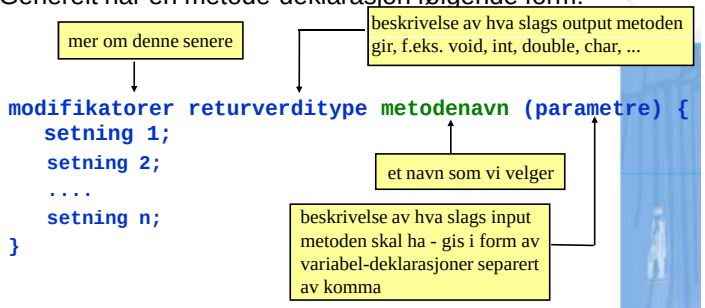
```
public static void main (String [] args) {
    .....
    .....
}
```

modifikatorer; retur-type; metode-navn; formelle parametre; metodekropp ("innmat")

- `static` brukes (nesten) ikke unntatt for `main`

Metode-deklarasjon

- Generelt har en metode-deklarasjon følgende form:



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 1: ingen parametre

- Følgende metode skriver ut en ordremeny på skjermen:

```
void skrivMeny () {
   System.out.println("Lovlige kommandoer: ");
   System.out.println("--------------------");
   System.out.println("1  Registrer ny student");
   System.out.println("2  Søk etter student");
   System.out.println("3  Lag liste");
   System.out.println("4  Avslutt");
   System.out.println("--------------------");
}
```

- Metodens navn: **skrivMeny**
- Returverditype **void** betyr at metoden ikke har noen returverdi
- Setningen **return;** kan plasseres hvor som helst i en metode med returverditype **void** og vil avbryte utførelsen av metoden

22

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

23

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**

24

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);
```

25

En liten digresjon: 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);
   }
}
```

Hva skjer når vi kjører det

```
class MittProgram {
   public static void main (String[] args) {
      Student s = new Student();
      s.skriv();
      System.out.println(”Programmet ferdig - ha det");
   }
}

class Student {
   String navn;
   student () {
      navn = ”Ola”;
   }
   void skriv() {
      System.out.println("Navnet mitt er:" + navn);
   }
}
```

```
Navnet mitt er:Ola
Programmet ferdig - ha det
```

28

Eksempel

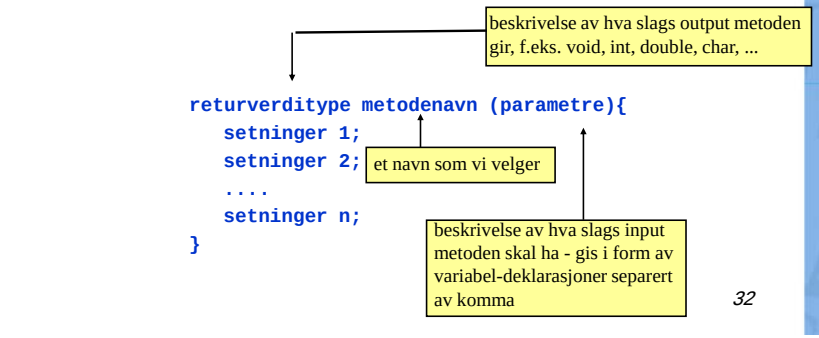
```
class StjerneProgram {

   public static void main (String[] args) {
      Stjerner stjerne = new Stjerner();
      stjerne.skrivStjerner();
      System.out.println("Hei");
      stjerne.skrivStjerner();
   }
}

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


Kompilering og kjøring

```
> javac StjerneProgram.java
> java StjerneProgram
**************************
**************************
**************************
**************************
Hei
**************************
**************************
**************************
**************************
```


Første oppsummering: metoder

- Java-programmene våre har så langt bestått av én klasse, startklassen, med én metode, main
- Fra nå: oppsett med to klasser og flere metoder i klassene
- Metodene vi ser på så langt har følgende form:



Levetiden til parametre og variable

- Vi kan ha adgang til tre typer variable i en metode:
  - **Objektvariable**: deklarert på klassenivå, inne i klassen, men utenfor metodene
  - **Lokale variable:** deklareres inni metoden, og er definert fra og med der deklarasjonen gjøres og til slutten av blokken de er deklarert i
  - **Parametre:** deklareres 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 samme objekt).

Eksempel

```
class Start {
   public static void main (String[] args) {
      Variabeltyper vt = new Variabeltyper();
      int intervall = 3;     // 'interval' er lokal variabel
      vt.økTid(intervall);
      vt.økTid(intervall);
   }
}

class Variabeltyper {
   int tid = 0;           // 'tid' er objektvariabel

   void økTid (int t) {   // 't' er parameter
      tid += t;
      System.out.println(tid);
   }
}
```

```
$ javac Start.java
$ java Start
3
6
$
```


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*.

Verdien til parmeterene 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 parameterene før setningene i metoden blir utført.***

Metode med returverdi

- Følgende metode leser et positivt tall fra terminal og returner det til kallstedet:

```
double lesPositivtTall () {
   In tastatur = new In();
   double x;
   do {
      System.out.print("Gi et positivt tall: ");
      x = tastatur.inDouble();
   } while (x <= 0);

   return x;
}
```

- Merk: vi kan hvor som helst i metoden gi instruksjonen

```
return <uttrykk>;
```

som avslutter utførelsen av metoden og returnerer til kallstedet med verdien til det angitte uttrykket (verdien må være av typen double i dette tilfellet).

Fullstendig eksempel

```
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
```

En klasse kan ha startmetode(r)

- Startmetoder heter det samme som klassen
- Et annet navn for en startmetode er en kontruktør
- Har ingen type (heller ikke void) foran deklarasjonen
- Kan ha parametre
- Kalles når man sier **new** på klassen (og lager et objekt)

```
import easyIO.*;

class MittProgram {
   public static void main (String[] args) {
      // her lager vi et objekt av den andre klassen
      // og kaller startmetoden
      Student s = new Student();
      s.skriv();
}}

class Student {
   String navn;
   Student () {       // startmetode i klassen Student
      navn = "Ola";
   }
   void skriv() {
         System.out.println("Navnet mitt er:" + navn);
}}
```

Metode med parameter og returverdi

- Følgende metode finner summen av elementene i en double-array:

```
double finnSum (double[] x) {
   double sum = 0.0;
   for (int i=0; i<x.length; i++) {
      sum += x[i];
   }
   return sum;
}
```


Eksempel på bruk

```
import easyIO.*;

class LengdeTest {
   public static void main (String[] args) {
      Out skjerm = new Out();
      Lengde l = new Lengde();
      double[] a = {2.3, 5.22, 3.6, 2.33, 8.6};
      double total = l.finnSum(a);
      skjerm.out("Samlet sum av a: ");
      skjerm.outln(total, 2);
   }
}

class Lengde {
   double finnSum (double[] x) {
      double sum = 0.0;
      for (int i=0; i<x.length; i++) {
         sum += x[i];
      }
      return sum;
   }
}
```

```
> java Lengde
Samlet lengde: 22.05
```

Metodekall

Anta at følgende eksekveres:

```
double [] a =
   {.....};
double total =
       l.finnSum(a);
```

Metoden som kalles:

```
double finnSum(double[] x) {
   double sum = 0.0;
   for (int i=0; i<x.length; i++) {
     sum += x[i];
   }
   return sum;
}
```

Eksekveringsrekkefølgen:

```
double total =
    l.finnSum(a);
```

argumentet lengde overføres

```
double[] x = a; // bare kopi av peker
double sum = 0.0;
for (int i=0; i<x.length; i++) {
      sum += x[i];
}
return sum;
```

uttrykket finnSum(a) gis verdien 22.05

```
total = 22.05;
```

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 deklareres 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;
}
```


Overlasting - eksempel

```
class Overlasting {
   public static void main (String[] args) {
      Skriv s = new Skriv();
      s.skrivUt(2,3);
      s.skrivUt(2.0,3.0);
      s.skrivUt(2.0,3);
   }
}

class Skriv {
   void skrivUt (double x, int y) {
      System.out.println("double+int: "+x + " , "+y);
   }
   void skrivUt (double x, double y) {
      System.out.println("double+double: "+x + " , "+y);
   }
}
```

double+int: 2.0 , 3
double+double: 2.0 , 3.0
double+int: 2.0 , 3

Oppgave 1: hva blir utskriften?

```
class Oppgave1 {

   public static void main (String[] args) {
      ToMetoder tm = new ToMetoder();
      System.out.println("Metode: main");
      tm.b();
   }
}

class ToMetoder {

   void a() {
      System.out.println("Metode: a");
   }

   void b() {
      a();
      System.out.println("Metode: b");
   }

}
```

```
> javac Oppgave1.java
> java Oppgave1
Metode: main
Metode: a
Metode: b
```

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
```

47

Parameteren i metoden main

- Vi kaller aldri direkte på metoden main (selv om det er lov) - det er Java-kjøresystemet som gjør dette når programmet starter.
- De argumenter vi gir etter `java ProgramNavn` blir overført til parameteren `String[] args` når main-metoden kalles.
- Eksempel:

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

      if (args.length == 0) {
         System.out.println("Ingen argumenter");
      }

      for (int i=0; i<args.length; i++) {
         System.out.print("Argument nr " + (i+1) + " var: ");
         System.out.println(args[i]);
      }

   }
}
```

```
> javac SkrivArgumenter.java
> java SkrivArgumenter
Ingen argumenter
> Java SkrivArgumenter a b c
Argument nr 1 var: a
Argument nr 2 var: b
Argument nr 3 var: c
```

Oppsummering om metoder

- Deklarasjon (lage metoden) :
  - Man pakker sammen de handlinger som hører sammen (gjør noe sammen) med krøllparenteser, og gir metoden et navn med vanlige parenteser bak navnet.
  - Man må også si om metoden returnere noe:
    - Returnerer **ingenting**: sett da `void` foren navnet
    - Returneren **en verdi**, sett *typen til verdien* foran navnet
      (Eks: `int, double, int[],..)`
      Metoden må da si `return XXX;` et sted i koden og hvor `XXX` er et uttrykk av den typen metoden skal returnere (eks `return i +14;)`.
  - Hvis metoden trenger noen data som den skal jobbe med for å gjøre 'jobben', settes de med type inn i parentesen bak navnet
    - Eks: `double kvadratrot(double x) {...; return ... }`
- Bruk / kall på metoden:
  - Man nevner navnet (i koden til en metode) med evt. parametere og med navnet på objektet b til klassen 'foran punktum':

```
y = 2.0 + b.kvadratrot(x*3.14);
```

49

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 (10 stk)

```
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: annulerKjøpAvFelt();         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("**AVSLUTTNING PÅ RURITANIAS OLJEFELTSYSTEM**");
   } // end ordreløkke

   // her deklarerer vi de 9 metodene:

} // end class Olje
```