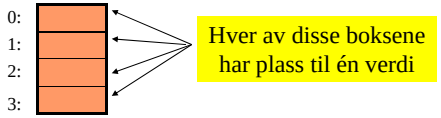
Inf1000 uke 4

Mer om arrayer

Metoder

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:



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];
```


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:

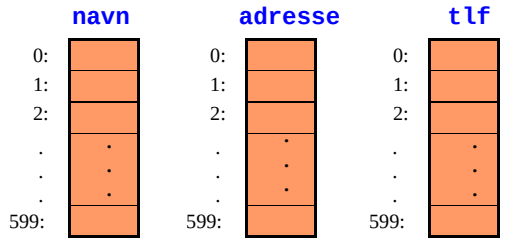
```
tlf.length // NB: ingen parenteser til slutt
```

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

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)

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"};
```

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: Arne
Alder: 60
Den yngste er Arild som er 40 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]);
        }

    }
}
```

13

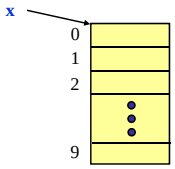
Resultat

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

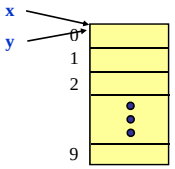
Hva skjedde?

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:

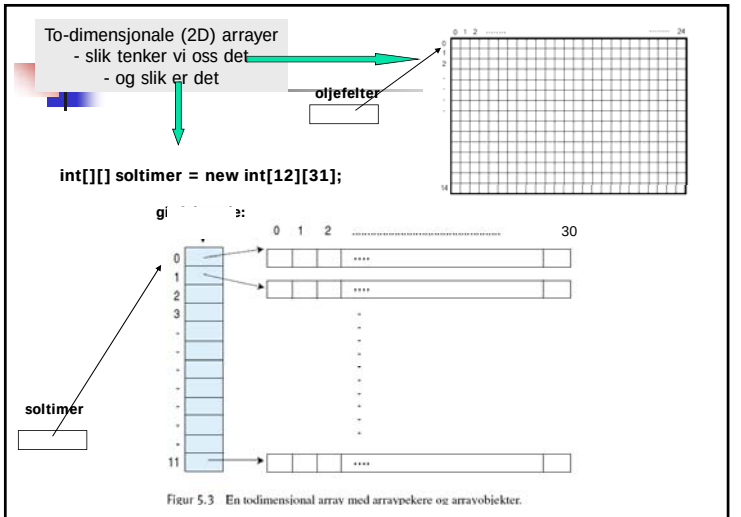
```
double[] y = new double[x.length];

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

16

2 dimensjonale arrayer (2D)

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

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

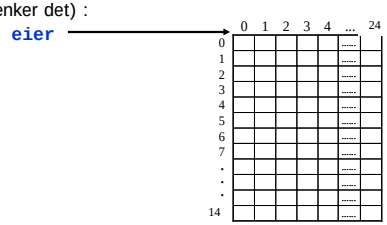
Figur 5.3 En todimensjonal array med arraypekere og arrayobjekter.

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
```

En klasse ***er*** noe - en metode ***gjør*** noe

- **Metoder**: Vi deler opp *handlingene* i programmet i metoder. En metode er da noen vanlige programsetninger som vi setter krøll-parenteser rundt. Metoden ***gjør*** det navnet på metoden sier. Vi velger selv navnet på de metodene vi lager.
  - EKS Banksystem: En metode for hver av handlingene: innskudd, uttak, beregnRenter, skrivRapport,…
- **Klasser**: Vi deler dataene og metodene i programmet opp i deler slik at hver av disse (klassene) tilsvarer en naturlig del av problemet:
  - EKS Banksystem: En klasse for hver av Banken, Kunde, Konto,…

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

Vi skal fra nå ha et **nytt oppsett** for programmer:
Minst to klasser, en med 'main' som starter den andre.

```
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
```

22

Blokker og metoder

- En blokk er en samling setninger omgitt av krøllparenteser:

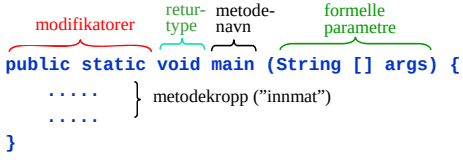
```
{
 setning 1;
 setning 2;
 ....
 setning n;
}
```

- Alle steder i et Java-program hvor det kan stå en setning, kan vi om ønskelig i stedet sette inn en blokk.
- Siden en blokk ofte forekommer flere steder i et program, hadde det vært praktisk om vi kunne definert blokken en gang for alle og gitt den et navn, slik at vi bare trengte å angi blokkens navn hvert sted vi ønsket å få utført setningene i blokken.
- Dette er fullt mulig i Java ved hjelp av det som kalles metoder.
- Vi skiller mellom å **deklarere** (lage/skrive) og **kalle** (bruke) en metode.

23

Metode-deklarasjon (lage metoden)

- En metode er essensielt en navngitt blokk med setninger som vi kan få utført hvor som helst i et program ved å angi metodens navn.
- Beskrivelsen av hva metoden skal hete og hvilke setninger som skal ligge i metoden kalles en metode-deklarasjon.
- main-metoden er et eksempel på en metode-deklarasjon:



- En klasse kan inneholde vilkårlig mange metode-deklarasjoner.
- `static` brukes (nesten) ikke, unntatt for `main`

24

Å deklarere en metode

- Generelt har en metode-deklarasjon følgende form:

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

Å benytte en metode

- Når vi benytter en metode sier vi at vi kaller på metoden.
- For å kalle på en metode uten parametre, skriver vi ganske enkelt som en setning:

```
metodenavn();
```

- For å kalle på en metode med parametre, må vi i tillegg oppgi like mange verdier som metoden har parametre, og i'te verdi må ha samme datatype som i'te parameter i metode-deklarasjonen. Eksempel:

```
metodenavn2(34.2, 53, 6);
```

- Hvis metoden returnerer en verdi, kan vi velge om verdien skal tas vare på eller ikke når metoden kalles. Eksempel på å ta vare på verdien:

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


Bruk av og kall på metoder

- En metode er et antall setninger som
  - gis et ***navn*** etterfulgt av en parentes ( ).
  - Etter parentesen kommer en klamme-parentes { } som omslutter setningene i metoden.
- Inne i parentesen kan det stå type og navn på ***parametre*** som er data som setningene inne i metoden kan bruke. (Når man *bruker* metoden, må man hver gang sette inn de verdier for disse parametrene som ønsker brukt av metoden.)
- Foran navnet står det minst ett ord som `int, void, double, String..` som sier hvilken type verdi som ***returneres*** (lages) av denne metoden
- Når man bruker en metode – ***'kaller***' en metode:
  - Hvis den returnerer en verdi, *kan* vi bruke kallet på metoden inne i et uttrykk på høyre side i en tilordningssetning
  - Hvis den ikke returnerer en verdi, skriver vi bare metodenavnet med de parameterverdier vi vil bruke i parentesen, etterfulgt av ; som en egen setning.
- Kall på en metode betyr at programmet 'hopper bort' til setningene i metoden, utfører disse setningene, og 'hopper tilbake' til rett etter der den ble kalt fra.

Metode uten parametre/returverdi

- 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("--------------------");
}
```

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

```
return;
```

som avslutter utførelsen av metoden og fortsetter utføringen av programmet til rett etter kallet.

Eksempel: metode uten input/output

- 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);
}
```

- Forklaring:
  - **void** er en returverditype som forteller at metoden ikke gir noe output.
  - skrivStjerner er det navnet vi har valgt å gi metoden

Eksempel på bruk

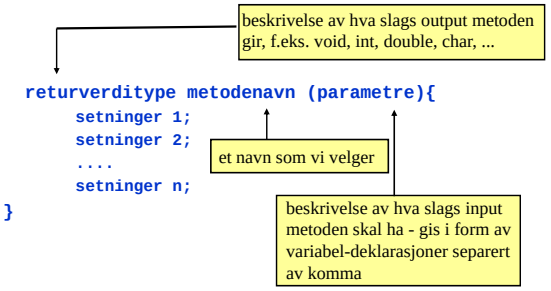
```
class Stjerner {

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

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

}
```


Kompilering og kjøring

```
> javac Stjerner.java
> java Stjerner
******************************
******************************
******************************
******************************
Hei
******************************
******************************
******************************
******************************
```


Første oppsummering: metoder

- Java-programmene så langt i kurset består av to klasser, startklassen med main og en annen klasse hvor det kan det befinne seg en eller flere metoder.
- De metodene vi ser på så langt i kurset har følgende form:



Levetiden til parametre og variable

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


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

- Når vi kaller metoden (bruker navnet i en annen metode), så overføres verdienene til de parameterene som ble brukt i kallet 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;

De verdiene som ble brukt 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

```
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

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

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]);
      }

   }
}
```

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 bare tilhører klassen, skrives `static` foran returtypen)
  - 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));
}}
```

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

   // her deklarerer vi de 9 metodene:

} // end class Olje
```

51