Inf1000 uke 5, 20. sept. 2005

Mer om arrayer
Metoder

Institutt for Informatikk
Universitet i Oslo

Arild Waaler

Mer om arrayer

Ofte kan vi bruke flere arrayer for å ordne informasjon av ulik type og bruke array-indeksen til å samordne informasjon i de forskjellige arrayene.

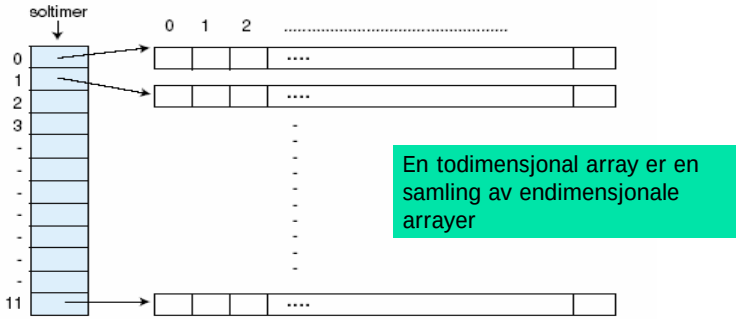
```
String[] månedsnavn = {"Januar", "Februar", "Mars", "April", "Mai",
                       "Juni", "Juli",  "August", "September", "Oktober",
                       "November", "Desember"};
```

Her vil indeksverdi 6 brukes til å lagre informasjon om juli-måned:

månedsnavn[6] gir navnet som en String.

Datastruktur for 2-dimensjonal array

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



En todimensjonal array er en samling av endimensjonale arrayer

```
antallMåneder = soltimer.length;     //antall måneder
antallDager   = soltimer[i].length; //antall dager i
                                     // den i-te måned
```


Initialisering og gjennomløp av arrayen

```
int[][] soltimer = {
  {2, 0, 3, 1, 0, 0, 4, 2, ... }, //soltimene for januar
  {3, 1, 0, 0, 4, 2, 2, 2, ... }, //soltimene for februar
  ...
  {2, 3, 1, 0, 0, 4, 2, 5, ... }  //soltimene for desember
};
```

Initialisering. En samling av verdier for en rad utgjør samlet sett én verdi for arrayen som grupperer radene (til kolonner)

```
// alle månedene behandles av denne løkka:
for (int mnd = 0; mnd < soltimer.length; mnd++) {
  int sum = 0;
  // alle dagene i inneværende måned behandles her
  for (int dag = 0; dag < soltimer[mnd].length; dag++) {
    sum = sum + soltimer[mnd][dag];
  }
  System.out.println("Soltimer i " +
                     månedsnavn[mnd] + " : " + sum);
}
```

Hver rad selekteres i tur i ytre løkke, hver verdi i raden behandles så i indre løkke

Tellevariabelen i for-løkka brukes til å hente frem navnet på måneden også.

Resultatet av kjøring

```
Soltimer i Januar : 14
Soltimer i Februar : 10
Soltimer i Mars : 20
...
```

Skal vi lagre denne informasjonen for hvert år, trenger vi en ekstra dimensjon for årstall. Da får vi en 3-dimensjonal array:

```
int[][][] soltimer = new int[10][12][31];
```


Eksempel: ta inn og ut av 2D-array

Vi skal lage et program som illustrerer hvordan man

- deklarerer og oppretter en to-dimensjonal array
- legger inn verdier i arrayen
- henter ut verdier fra arrayen

Programmet skal:

- be om og lese inn ialt 3*4 = 12 heltall og legge tallene inn i en int-array med 3 rader og 4 kolonner
- skrive ut innholdet av int-arrayen

Det ferdige programmet

```
import easyIO.*;

class ArrayEksempel {
   public static void main (String[] args) {
      In tast = new In();
      Out skjerm = new Out();
      int[][] a = new int[3][4];

      for (int i=0; i<3; i++) {
         for (int j=0; j<4; j++) {
            skjerm.out("Gi et heltall: ");
            a[i][j] = tast.inInt();
         }
      }

      for (int i=0; i<3; i++) {
         for (int j=0; j<4; j++) {
            skjerm.out(a[i][j], 5);
         }
         skjerm.outln();
      }
   }
}
```


Eksempel: liste over ulike verdier i 2D-array

Vi skal lage et program som illustrerer hvordan man

- løper gjennom et 2D-array og ser på alle verdier
- legger inn i en (en-dimensjonal) array de ulike verdiene som forekommer i 2D-arrayen
- skriver ut listen med de ulike verdiene

Programmet skal:

- be om og lese inn verdier til 2D-arrayen
- skrive ut en sortert liste over de entydige verdier som forekommer i arrayen

Programskisse

```
import easyIO.*;

class ArrayEksempel2 {
   public static void main (String[] args) {
      In tast = new In();

      int[][] a = new int[3][4];

      for (int i=0; i<3; i++) {
         for (int j=0; j<4; j++) {
            System.out.print("Gi et heltall: ");
            a[i][j] = tast.inInt();
         }
      }

      <lag liste over entydige verdier som forekommer
         i arrayen a, og skriv ut listen>

   }
}
```


Å lage liste over entydige verdier

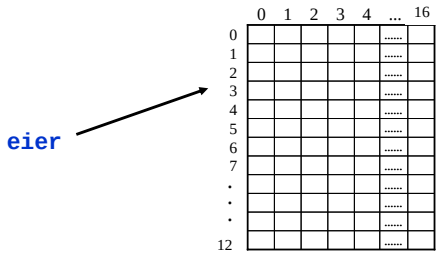
```
int[] verdier = new int[12];
int antall = 0;

for (int i=0; i<3; i++) {
   for (int j=0; j<4; j++) {
      int aij = a[i][j];
      boolean funnet = false;

      for (int k=0; k<antall; k++) {
         if (aij == verdier[k]) {
            funnet = true;
         }
      }

      if (!funnet) {
         verdier[antall] = aij;
         antall++;
      }
   }
}

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


Oblig 2 bruker 2D-array

- Oljefelt-tabellen deklareres som en array der

  `String[][] eier = new String[13][17];`

  Vi kan visualisere en 2D-array som en tabell:



Hvis et felt ikke er solgt:
`eier[i][j] = = null`

Hvis et felt er solgt:
`eier[i][j] ! = null`

- Eksempler på lovlige operasjoner:

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


Eksempel: finn antall solgte oljefelt

```
class AntallFelt {
  public static void main (String [] args) {
    String[][] eier = new String[13][17];
    <innlesning av eiere m.m.>

    int antallSolgte = 0;
    for (int i=0; i<13; i++) {
      for (int j=0; j<17; j++) {
       if (eier[i][j] != null) {
          antallSolgte++;
       }
      }
    }
    System.out.println("Antall solgte felt: " + antallSolgte);
  }
}
```

Et java-program består av klasser og metoder

- Et stort program må deles opp i mindre deler for:
  - Å greie å få det riktig (sjekke/teste hver del for seg)
  - Dele det opp i logiske biter (ha sammen det som logisk hører sammen)
- Det er to måter man kan dele opp programmet på i et Objekt-orientert programmerings-språk som Java:
  - **Metoder**: Slå sammen noen setninger (tilordninger, while- og for-løkker, deklarasjoner av noen variable) og gi disse et navn.
  - **Klasser:** Dele opp programmet i de 'store' delene det består av. En klasse *inneholder* metoder og deklarasjoner og har et navn (eks: class Bankkunde {..})
- Vi ønsker en oppdeling som gjør det lettere å programmere:
  - Metoder 'lager' store instruksjoner som er enkle å bruke (eks: sort(), sqrt())
  - Klasser deler *hele* problemet opp i håndtérbare deler.
  - **Splitt og hersk !**

Enklere å løse Oblig2 med 9 nye metoder

```
import easyIO.*;

class Oblig2 {

  static In tast = new In();

  public static void main (String[] args) {

    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: finnGjennomsnittUtvinning(); break;
            default: break;
        } // end switch

    }// end while flere kommandoer

    System.out.println("**AVSLUTTNING PÅ RURITANIAS OLJEFELTSYSTEM**");
  } // end main

  // her deklarerer vi de 9 metodene: skrivMeny, , ...finnGjennomsnittUtvinning

} // end class Oblig2
```


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

Metoder i én klasse skal vi lære idag

Klasser, objekter og metoder i flere klasser skal vi lære de to neste ukene

Blokker og metoder

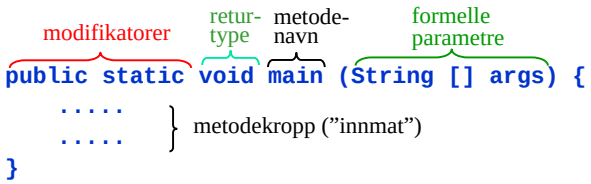
- En blokk er en samling instruksjoner omgitt av krøllparenteser:

```
{
 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 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 instruksjonene i blokken.
- Dette er fullt mulig i Java ved hjelp av det som kalles metoder.

Metode-deklarasjon (lage metoden)

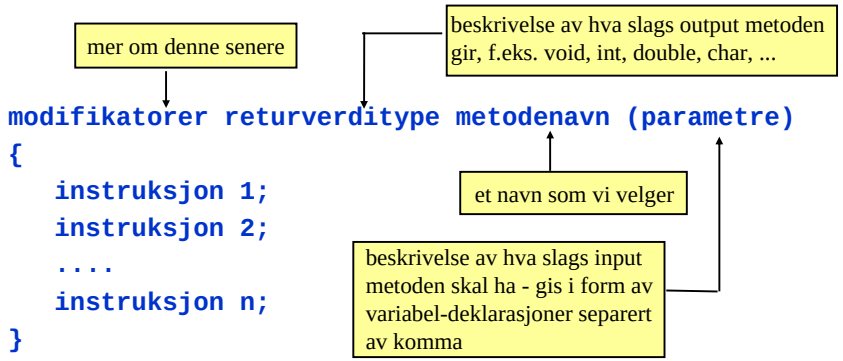
- En metode er essensielt en navngitt blokk med instruksjoner som vi kan få utført hvor som helst i et program ved å angi metodens navn.
- Beskrivelsen av hva metoden skal hete og hvilke instruksjoner som skal ligge i metoden kalles en metode-deklarasjon.
- main-metoden er et eksempel på en metode-deklarasjon:



- En klasse kan inneholde vilkårlig mange metode-deklarasjoner.
- `static` bruker vi fordi hittil ***ikke*** har lært å lage objekter av klasser.
- `static` skal **fjernes** fra (nesten) alle metoder unntatt `main` når vi har lært å lage objekter.

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

```
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:

```
metodenavn(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 = metodenavn(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.
- 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:

```
static 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:

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

- Forklaring:
  - **`static`** er en modifikator som forteller at dette er en klassemetode og ikke en objektmetode, dvs metoden skal ikke benyttes inni et objekt.
  - **`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

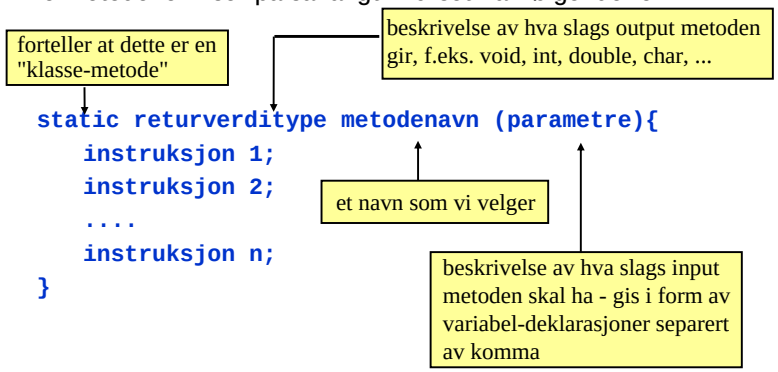
```
class Stjerner {

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

  static 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 en enkelt klasse, og i klassen kan det befinne seg en eller flere metoder (en av disse må hete main).
- De metodene vi ser på så langt i kurset har følgende form:



Metode med returverdi

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

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

   return x;
}
```

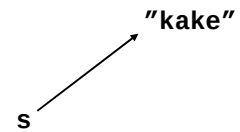
- Merk: vi kan hvor som helst i metoden gi instruksjonen

```
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).

Metoder i klassen String

- En tekststreng er en sekvens av tegn (null, en eller flere), f.eks.
  - `""`
  - `"&"`
  - `"Kaia er student"`
- Hver tekststreng vi lager er et *objekt* av typen String.
- En String-variabel (f.eks. `String s`) er en *referanse* til et slikt objekt
- Resultatet av å utføre `String s = "kake"` :



- For å finne lengden (dvs antall tegn i) en tekst:

```
int lengde = s.length()
```

Merk at s "eier" metoden length(). Dette vi du høre mer om når vi kommer til objekter.

equals-metoden for strenger

- For å teste om to tekststrenger er like, brukes metoden equals:

```
// Anta at s og t er tekstvariable (og at s ikke har verdien null)
if (s.equals(t)) {
   System.out.println("Tekstene er like");
} else {
   System.out.println("Teksten er forskjellige");
}
```

- Bruk av == virker av og til, men ikke alltid:

```
String s = "abc";
String t = "def";
String tekst1 = s + t;
String tekst2 = s + t;
```

Nå er `tekst1.equals(tekst2)` true, mens `tekst1 == tekst2` er false.

Merk at kall på tekst1.equals() gir en Boolean-verdi siden returverdien til equals er Boolean.

Parametre og argumenter

```
class Eksempel {

   public static void main (String[] args) {
      minMetode(3.14, 365);
   }

   static 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 oveføres verdienene til de parameterene som ble brukt i kallet slik:

```
public static void main (String[] args) {
                int i = 17;
                   minMetode(3.14, i +2);
 }
                                  x = 3.14;
                                  y = i+2;
 static void minMetode (double x, int y)   {
    // nå kan x og y brukes med de verdier de fikk i kallet
                   .....
 }
```

*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:

```
static 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 LesPositivtTall {
   public static void main (String[] args) {
     Out skjerm = new Out();
     double x = lesPositivtTall();
     double y = lesPositivtTall();
     skjerm.out("ln(x*y) = ");
     skjerm.outln(Math.log(x*y), 2);
   }
   static 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
```

Metode med parameter og returverdi

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

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

 static 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 [] lengde =
   {.....};
double total =
       finnSum(lengde);
```

Metoden som kalles:

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

Eksekveringsrekkefølgen:

```
double total =
    finnSum(lengde);
```

argumentet lengde overføres

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

uttrykket finnSum(lengde) 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};
      finnDelsummer(a);
      System.out.println("a[3] = " + a[3]);
   }

   static 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:

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

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


Overlasting - eksempel

```
public static void main (String[] args) {
    skrivUt(2,3);
    skrivUt(2.0,3.0);
    skrivUt(2.0,3);
}

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

static 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) {
    System.out.println("Metode: main");
    b();
  }

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

  static 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) {
    int x = 1;
    while (g(x) > 0) {
      System.out.println(x++);
    }
  }

  static 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:
    `y = 2.0 + kvadratrot(x*3.14);`