INF1000 (Uke 6)
Metoder

Grunnkurs i programmering

Institutt for Informatikk

Universitetet i Oslo

Are Magnus Bruaset og Arild Waaler

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

Metoder

- 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.

  Da trenger vi bare 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

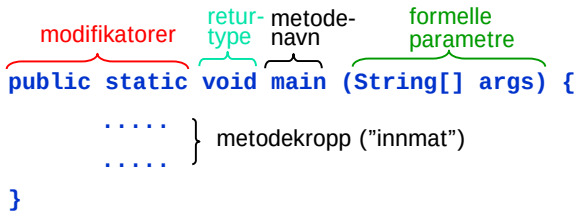
Metoder

- 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.

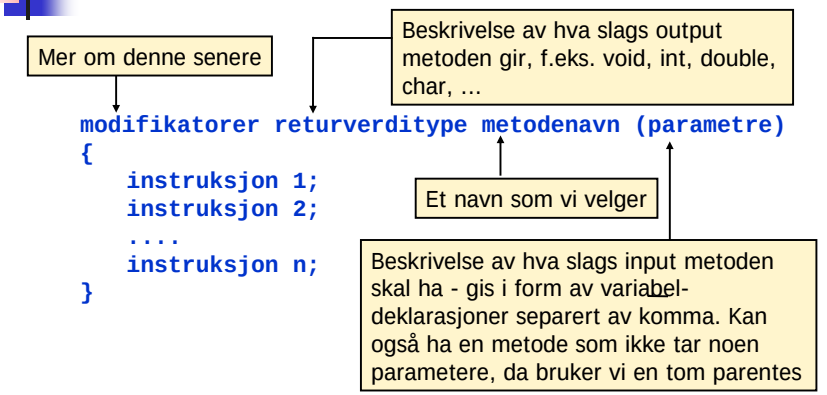
Metoder

- **`main`**-metoden er et eksempel på en metode-deklarasjon:



- En klasse kan inneholde vilkårlig mange metode-deklarasjoner

Å deklarere en metode



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 parametere, skriver vi ganske enkelt

```
metodenavn();
```


Metoder med parametre

- Ved kall på en metode med parametere må
  - vi oppgi like mange verdier som metoden har parametere
  - i'te verdi må ha samme datatype som i'te parameter i metode-deklarasjonen
- Eksempel:

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

Metoder med returverdi

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

```
double sum = metodenavn(25.3, 52, 7);
```


Eksempel: Metode uten input/output

```
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();
  } //slutt main

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

} //slutt Stjerner
```


Kompilering og kjøring

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

3 typer variable: Klassevariable

- Klassevariable
- Lokale variable
- Parametere

Klassevariable

- Variable som er deklarert på klassenivå, utenfor metoden
- (Også objektvariable)

Lokale variable

- Variable som deklareres inne i en metode
- Slike variable er definert fra og med der deklarasjonen gjøres og til slutten av blokken de er deklarert i

Parametere

- Variable som deklareres i hodet på metoden
- Slike variable er definert i hele metodekroppen

Viktig detalj

Ved gjentatte kall på en metode lages det et

nytt sett med lokale variable og parametere

hver gang

Eksempel

```
class Variabeltyper {
   static int tid = 0;          // Klassevariabel

   public static void main (String[] args) {
      int intervall = 3;        // Lokal variabel
      økTid(intervall);
      økTid(intervall);
   }

   static void økTid (int t) {  // Parameter
      tid += t;
      System.out.println(tid);
   }

}
```


Metode uten parametere og returverdi

Følgende metode skriver ut en ordremeny på skjermen:

```
static void skrivMeny () {
   System.out.println("Velg: ");
   System.out.println("--------------------");
   System.out.println("1  Cappuccino ");
   System.out.println("2  Cafe Latte");
   System.out.println("3  Americano ");
   System.out.println("4  Espresso ");
   System.out.println("--------------------");
}
```


Parametere og argumenter

```
class Eksempel {

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

   static void minMetode (double x, int y) {
      .....
   }

}
```

Argumenter

Parametere

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

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: instruksjonen

return <uttrykk>;

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("Du har lest inn x = " + x);
      skjerm.out(" og y = " + y + ", ln(x*y) = ")
      skjerm.outln(Math.log(x*y), 2);
   } //avslutter main
   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; //her blir x returnert til der metoden kalles fra
   } //avslutter lesPositivtTall
} //avslutter LesPositivetTall
```


Metode med parameter og returverdi

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

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


Metodekall

Anta at følgende eksekveres:

```
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;
}
```

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);
    } //her slutter main

   static double finnSum (double[] x) {
       double sum = 0.0;
       for (int i = 0; i < x.length; i++) {
           sum += x[i];
       }
       return sum; //her returneres sum til der metoden er kalt
fra.
    } //her avsluttes metoden finnSum
} //her avsluttes klassen Lengde
```


Rekkefølge i eksekvering

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

Argumentet **lengde** overføres til variabelen x i metoden finnSum

```
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 referanser som parametere

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 til høyre 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];
      }
   }

}
```


Overlasting av metoder – Et eksempel

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


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

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 parametere eller
  - metodene har ulik type på noen av parametrene, slik at Java alltid finner en entydig match

Eksempel

- Overlasting:

```
static int skrivUt(double x, int y) {...}
static int skrivUt(double x, double y) {...}
```

Her kan vi f.eks. ha kallet `skrivUt(2,7)`
- da velges første metode

Eksempel

- Overlasting:

```
static int skrivUt(double x, int y) {...}
static int skrivUt(int x, double y) {...}
```

Her får vi kompilatorfeil hvis vi forsøker kallet `skrivUt(2,7)` !

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 (`main`)

```
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]);
      }
   } //her avsluttes main
} //her avsluttes SkrivArgumenter
```


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");
   }
}
```


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;
   }

}
```


Tekster og klassen `String`

- En tekststreng er en sekvens av tegn (null, en eller flere), f.eks.

  `""`

  `"*"`

  `"Kristina"`

- Hver tekststreng vi lager er et *objekt* av typen **`String`**

Konkatenering

- Operatoren + har flere betydninger i Java:
  - mellom to tall: addisjon
  - mellom to tekster : tekstkonkatenering
  - mellom tekst og annen type : tekstkonkatenering
- Eksempel på overlasting av metode

Eksempel

Husk at uttrykk i Java beregnes fra venstre mot høyre:

```
class Konkatenering {
   public static void main (String[] args) {
      System.out.println("Sum: " + 2 + 3);
      System.out.println(1 + 2 + 3 + " " + 1 + 2 + 3);
   }
}
```

```
>java Konkatenering
Sum: 23
6 123
```


Konvertere mellom små og store bokstaver

- Vi kan konvertere fra små til store bokstaver:

```
String s = "Jeg ER 18 år";
String s2 = s.toUpperCase();
// Nå er s2 tekststrengen "JEG ER 18 ÅR"
```

- Vi kan konvertere fra store til små bokstaver:

```
String s = "Jeg ER 18 år";
String s2 = s.toLowerCase();
// Nå er s2 tekststrengen "jeg er 18 år"
```

- Det finnes tilsvarende metoder for å konvertere char-verdier:

```
char c = 'x';
char c2 = Character.toUpperCase(c);
char c3 = Character.toLowerCase(c);
```

NB: merk skrivemåten!

Eksempel 1

- Metode som lager stor forbokstav i en tekststreng:

```
static String StorForbokstav (String s) {
   String t;
   if (s.length() > 0) {
      char c = Character.toUpperCase(s.charAt(0));
      t = c + s.substring(1);
   } else {
      t = "";
   }
   return t;
}
```

Alfabetisk ordning

- Anta at **s** og **t** er tekstvariable (og at **s** ikke har verdien null)
- Er **s** foran **t** i alfabetet?

```
int k = s.compareTo(t);

if (k < 0) {
  System.out.println(s + " er alfabetisk foran " + t);
} else if (k == 0) {
  System.out.println(s + " og " + t + " er like");
} else {
  System.out.println(s + " er alfabetisk bak ” + t);
}
```


Inneholder en tekst en annen?

- Anta at **s** og **t** er tekstvariable (og at **s** ikke har verdien null)
- Inneholder **s** teksten **t**?

```
int k = s.indexOf(t);

if (k < 0) {
  System.out.println(s + " inneholder ikke " + t);
} else {
  System.out.println(s + " inneholder " + t);
  System.out.println("Posisjon til" + t + " i " + s + " er " + k);
}
```


Starter en tekst med en annen?

- Anta at **s** og **t** er tekstvariable (og at **s** ikke har verdien null)
- Starter **s** med teksten **t**?

```
boolean b = s.startsWith(t);

if (b) {
   System.out.println(s + " starter med " + t);
} else {
   System.out.println(s + " starter ikke med " + t);
}
```


Slutter en tekst med en annen?

- Anta at **s** og **t** er tekstvariable (og at **s** ikke har verdien null)
- Slutter **s** med teksten **t**?

```
boolean b = s.endsWith(t);

if (b) {
  System.out.println(s + " ender med " + t);
} else {
  System.out.println(s + " ender ikke med " + t);
}
```

Fra tall til tekst og omvendt

- For å konvertere fra tall til tekst:

```
String s1 = String.valueOf(3.14);
String s2 = String.valueOf('a');
String s3 = String.valueOf(false);

String s4 = "" + 3.14
String s5 = "" + 'a';
String s6 = "" + false;
```

- For å konvertere fra tekst til tall:

```
int k = Integer.parseInt(s);
double x = Double.parseDouble(s);
//(og tilsvarende for de andre numeriske datatypene)
```


Å finne enkeltord i en tekst

- Av og til ønsker vi å kunne bryte opp en tekst i de enkelte ordene, der ordene er separert av spesielle skilletegn
- String metoden **`split(…)`** er et verktøy som kan brukes til dette

Mer om for-løkke

- Anta at vi har en array ord av typen `String` som vi ønsker å gå gjennom en gang.
- Sist så vi på for løkker av formen:

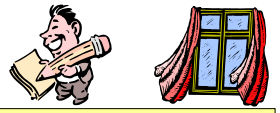
```
for (int i = 0; i < ord.length(); i++){
}
```

- En annen måte å gå gjennom en tabell ved hjelp av for-løkke er:

```
for (String s : ord){
}
```

- Begge løkkene vil gå gjennom hvert element i en arrayene systematisk fra første til siste element.
- For å bruke kommandoen `for (String s : ord)` må du kjøre Java 1.5, gamle versjoner av java har ikke denne kommandoen.

Eksempel



```
import easyIO.*;

class SplitDemo {
  public static void main(String [] args){
    In tast = new In();
    String mønster = " ";
    System.out.print("Skriv en setning: ");
    String linje = tast.inLine();
    String[] ord = linje.split(mønster);

    for (String s: ord) {
      System.out.println(s);
    }
  }
}
```

Regulære uttrykk

Tegn	Betydning
.	alle tegn
\d	siffer (0-9)
\D	alt som ikke er siffer
\s	blanke
\S	alle ikke blanke tegn
\w	siffer og alle bokstaver i det engelske alfabet
\W	Alt som ikke er siffer eller bokstaver i det engelske alfabet

Når regulære uttrykk brukes som mønster vil teksten splittes hver gang et av tegnene listet under **betydning** forekommer. Disse tegnene vil ikke være med i tekst-arrayen som blir generert av `split(String s)` metoden i klassen String.

Eksempel Andre skilletegn

```
import easyIO.*;

class SplitDemo2 {
  public static void main(String [] args){
    In tast = new In();
    System.out.print("Skriv inn mønster: ");
    String mønster = tast.inLine;
    System.out.print("Skriv en setning: ");
    String linje = tast.inLine();
    String[] ord = linje.split(mønster);
    for (String s: ord) {
      System.out.println(s);
    }
  }
}
```