



# INF1000 (Uke 6)

## Mer om metoder, tekster

---

Grunnkurs i programmering

Institutt for Informatikk

Universitetet i Oslo

Anja Bråthen Kristoffersen og Are Magnus Bruaset



# Orakeltjeneste på Abel

## Hjelp til Obligatorisk oppgave 2

---

- Tirsdag 22. feb.: 10 - 14
- Onsdag 23. feb.: 14 – 19

Sted: terminalstuen på Abel

(underetasjen i Niels Henrik Abels hus)



# Rep: Metoder

---

- Java-programmene så langt i kurset:
  - består av en enkelt klasse
  - i klassen kan det befinne seg en eller flere metoder (en av disse må hete **main**)

# Rep: Metoder

Forteller at dette er en "klasse-metode"

Beskrivelse av hva slags output metoden gir, f.eks. void, int, double, char, ...

**static returverditype metodenavn(type par1, type par2, ..)**

{

**instruksjon 1;**  
**instruksjon 2;**

...

**instruksjon n;**

}

Et navn som du velger

Beskrivelse av hva slags input metoden skal ha – null en eller flere parametere, gis i form av variabel-deklarasjoner separert med komma



# 3 typer variable: Klassevariable

---

- Klassevariable
- Lokale variable
- Parametere



# Klassevariable

---

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



# Lokale variable

---

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



# Parametere

---

- Variable som deklarereres 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
    } //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 deklarereres 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.

`""`

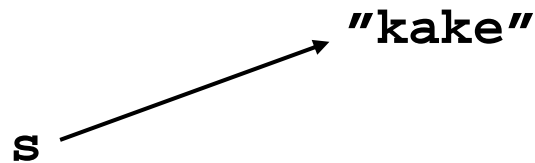
`"*"`

`"Kristina"`

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

# Tekster og klassen `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();
```



# Bruk av spesialtegn

---

- Både i **char**-uttrykk og i **String**-uttrykk kan vi ha mange ulike typer tegn
- Alle Unicode-tegn er tillatt
- Unicode er en standard som tillater tusenvis av tegn (ulike varianter fins; den som støttes av Java tillater 65536 ulike tegn)



# Unicode-tegn i Java

---

- Alle tegnene kan angis som `'\uxxxx'` hvor hver x er en av  
0, 1, 2, ..., 9, A, B, C, D, E, F

Eksempel: `'\u0041'` er tegnet 'A'

- Noen spesialtegn har egen forkortelse:
  - `\t` tabulator
  - `\n` linjeskift
  - `\"` dobbelt anførselstegn
  - `\'` enkelt anførselstegn
  - `\\` bakslask



# 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
```



# Teste om to tekster er like

---

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

```
String s = "abc";  
String t = "def";  
System.out.println((s+t) == (s+t));  
// Setningen over skriver ut false
```



# Teste om to tekster er like

---

- For å teste om to tekststrenger er like, brukes 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");
}
```



# De enkelte tegnene i en streng

- Tegnene i en tekststreng har posisjoner indeksert fra 0 og oppover

0	1	2	3
'k'	'a'	'k'	'e'

- Vi kan få tak i tegnet i en bestemt posisjon:

```
String s = "kake";  
char c = s.charAt(1);  
// Nå er c == 'a'
```



# De enkelte tegnene i en streng

---

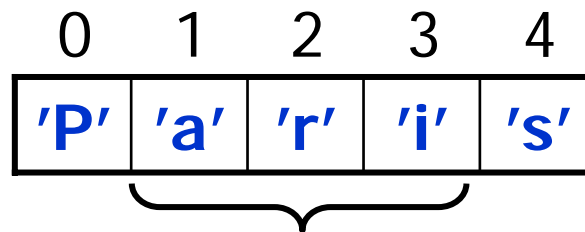
- Vi kan erstatte alle forekomster av et tegn med et annet tegn:

```
String s1 = "kake";  
String s2 = s1.replace('k', 'r');  
// Nå er s2 en referanse til tekststrengen "rare"
```

# Deler av en tekststreng

- Vi kan trekke ut en del av en tekststreng:

```
String s = "Paris";  
String s1 = s.substring(1,4);  
// Nå er s1 tekststrengen "ari"
```



`s.substring(1,4)`

# Deler av en tekststreng

- Generelt:

`s.substring(index1, index2)`

Første posisjon som  
skal være med



Første posisjon som  
*ikke* skal være med

- Siste del av en tekststreng:

```
String s = "Paris er hovedstaden i Frankrike";  
String s1 = s.substring(6);  
// Nå er s1 tekststrengen "er hovedstaden i Frankrike"
```

# 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

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

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

# 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);
        }
    }
}
```